REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	30 APR 1997	**Addendum to Final Technical Report (Sep '96 - Apr '97)**

4. TITLE AND SUBTITLE
Addendum to Final Technical Report
Computer Aided Performance Engineering: Current State of the art

5. FUNDING NUMBERS
C - DAAH01-96-C-R298
PR - PAN RTW X8-96

6. AUTHORS
Ramesh M. Reddi **Infopike, Inc.**
Prof. Reda Ammar **Univ of Connecticut**

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Infopike, Inc., P. O. Box 328, Norwich, CT 06360
University of Connecticut, Storrs, CT 06269

8. PERFORMING ORGANIZATION REPORT NUMBER

8. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, VA 22203-1714

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This is an addendum to the Final Technical Report submitted as a result of Phase I SBIR work awarded to Infopike, Inc. This addendum reflects the work done by us in assessing the state of the art in the field of Computer Aided Performance Engineering.

Performance evaluation of computer and software systems has become a rapidly growing field with a growing number of tools being developed for analyzing various performance aspects of such systems. The literature on performance evaluation methodologies has also mushroomed with various proposals from researchers all over the world. This addendum presents the results of a survey conducted on automated performance analysis tools for computer systems.

The survey includes measurement-based tools, analytical tools, simulation tools and visualization tools, and describes their properties and capabilities. The tools have been categorized based on their analysis capabilities and include system-oriented, process-oriented and module-oriented categories. The tools surveyed in this paper incorporate various techniques including simulation, modeling (Petri net, queuing, semi-markov etc.), measurement, visualization and emulation. We also present a new methodology for tool classification that aids system designers.

14. SUBJECT TERMS
Computer Aided Performance Engineering
Performance (analytical, simulation, measurement, visualization) tools

15. NUMBER OF PAGES
38

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	**UNCLASSIFIED**	**UNCLASSIFIED**	**SAR**

Computer Aided Performance Engineering : Current state of the art

Reda A Ammar, Nagarajan Kandasamy, Brian Mackay and Howard A Sholl.
U-155, Computer Science and Engineering Department,
University of Connecticut,
Storrs, CT 06269.

Ramesh M Reddi,
Infopike Inc.,
Norwich, CT 06360.

Performance evaluation of computer and software systems has become a rapidly growing field with a growing number of tools being developed for analyzing various performance aspects of such systems. The literature on performance evaluation methodologies has also mushroomed with various proposals from researchers all over the world. This paper presents the results of a survey conducted on automated performance analysis tools for computer systems. The paper also surveys some of the evaluation methodologies proposed by various authors. The survey includes measurement based tools, analytical tools, simulation tools and visualization tools, and describes their properties and capabilities. The tools have been categorized based on their analysis capabilities and include system-oriented, process-oriented and module-oriented categories. The tools surveyed in this paper incorporate various techniques including simulation, modeling (Petri net, queuing , semi-markov etc.), measurement, visualization and emulation.. The vast number of tools available to developers of computer systems makes selection of the appropriate tool an increasingly difficult task. This paper presents a new methodology for tool classification which the authors hope will capture the characteristics of individual tools better than the standard table format. This model will also make tool search based on specific characteristics easier for the designers who need the appropriate tools.

Keywords : computer aided performance engineering, performance, (analytical, simulation, measurement, visualization) tools.

1 INTRODUCTION :

The number of tools that have been developed for computer system performance analysis is overwhelming. The underlying analysis methodology, system requirements (hardware and software), analysis capabilities and front ends makes each tool unique. This paper surveys some of the tools available both in the industry and research community and discusses some analysis methodologies proposed in the literature.

Performance of a computer system can be estimated by one of the three broad categories, namely : analytically, through simulation, or by measurement. The analytical approach focuses on building a static model of the system under study. The resulting model describing the behavior of the system, can then be solved to obtain performance estimates. The input parameters to the model affect the estimates obtained. It is important to note that this approach focuses on solving a static model of the underlying system. In contrast to this are the simulation and measurement based approaches. These approaches focus on the dynamic behavior of the system. Simulation tools require that the user write the program in a high level simulation language and provide system characteristics to the tool. Simulation tools provide developers with a virtual machine on which to execute their code and get performance estimates. Thus estimates for a target architecture can be obtained

by simulating the code on the development machine. Measurement tools do not require that a model be generated as the performance data is obtained directly from the underlying system. This requires instrumentation of the code at various levels. The techniques outlined above have their advantages and disadvantages. Analytical modeling becomes increasingly intractable as the complexity of the system increases. The demands made by simulation tools on processor time and memory increase as the complexity of the model increases. Measurement tools must be as non-intrusive as possible so as not to influence the performance data obtained. The difficulties with the techniques has led some researchers to move towards hybrid models involving the three techniques. The authors would like to point out that most of the tools surveyed in the paper use the simulation or measurement approaches.

In addition to categorizing the tools as analytical, simulation, or measurement, it is also possible to describe the tool in terms of its analysis capabilities. Several of the tools surveyed in this paper exhibit analysis hierarchies. A computer system can be analyzed at different levels. These levels are incorporated into an hierarchy that has the system as a single entity at the highest level and individual processes (user programs) at the lowest. Individual processes are considered to be composed of a number of modules and analysis at the module level focuses specifically on the performance of a single module.

For the purposes of this paper, we classify the hierarchy into three levels namely system level, process level and module level. A brief description of the terminology used is presented below :

- *System level* : This represents the highest level of abstraction, and includes all components of a computer system including hardware interfaces to external sources. Analysis at this level is of large granularity and focuses primarily on the performance of the system in terms of throughput, waiting time etc.
- *Process level* : Performance Analysis at this level focuses on the process itself and in the case of parallel and distributed systems, interaction among the processes constituting such a system. A number of process hierarchies can exist depending on the complexity of the software system. This level can also be called as the job level where a job comprises of one or more processes. The operating system is also treated as a job on the system and can be analyzed at this level.
- *Module level* : Analysis at the *module level*, the lowest level in the hierarchy, is focused primarily on the modules (procedures and/or functions) of the individual processes.

The tools surveyed in this paper can be categorized in terms of these hierarchy levels. Many of the tools surveyed exhibit capabilities that would put them in more than one category of analysis.

2 SYSTEM LEVEL ANALYSIS :

2.1 Simulation Techniques:

Introduction to simulation :

A computer system's performance can be evaluated by various simulation techniques such as *emulation, Monte Carlo, trace-driven and discrete event simulation.* The simulation approach can be used to analyze complex systems which are difficult to measure and model using analytical techniques. An overview of the various simulation techniques introduced above follows :

Monte Carlo Simulation :

A static simulation or one without a time axis is called a *Monte Carlo* simulation. Such simulations are used to model probabilistic phenomenon that do not change characteristics with time. These simulations require the generation of pseudo-random numbers. *Monte Carlo* simulations are also used for evaluating non-probabilistic expressions using probabilistic methods.

Trace-Driven Simulation :

A simulation using a trace as its input is a *trace-driven simulation.* A trace is a time-ordered record of events on a real system.. Trace-driven simulations are quite common in computer system analysis. They are generally used in analyzing or tuning resource management algorithms. Trace-driven simulation has been used to analyze various algorithms including paging algorithms, cache analysis, CPU scheduling algorithms, deadlock prevention algorithms, and algorithms for dynamic allocation of storage. A trace of the resource demand is used as an input to the simulation, which models different algorithms. For example,

in order to compare different memory management schemes, a trace of page reference patterns of key programs can be obtained on a system. This trace can then be used to find the optimal set of parameters for a given memory management algorithm or to compare different algorithms.

Discrete-Event Simulations :

A discrete-event model represents a process in which the system state changes in distinct steps. These state changes are usually characterized by the passage of time. Systems that can be described by discrete-event models are those in which resource contention and allocation occurs. Queuing and probabilistic behavior are important phenomena encompassed by discrete-event models. Computer systems exhibit such behavior and are excellent subjects for *discrete-event simulation.*

All discrete-event simulations have a common structure regardless of the system being modeled. If a general-purpose language is used, all the components have to be developed by the analyst. A simulation language provides some of the components and leaves others for the analyst to develop. Common components provided by such languages include an *event scheduler, simulation clock and a time-advancing mechanism, system state variables, event routines, input routines, report generator, initialization routines, trace routines, dynamic memory management and the main program.*

There are three approaches to developing a discrete-event simulation : the event-oriented approach, the process-oriented approach, and the activity-oriented approach. For the event-oriented approach, the model is described by a series of events between which simulated time may elapse. An event usually changes the state of the system. Using the process-oriented approach, the model is described by a number of interacting processes which can represent either independent procedures, where a procedure is a sequence of activities (sometimes referred to as the *transaction-oriented view*) or resources (sometimes referred to as the *resource-oriented view*). A simulation using the activity-oriented approach is defined by the number of activities which are executed when certain conditions are met. Simulation time advances in increments, and at each advance the activity list is checked. All activities scheduled to execute at a particular time are executed.

Simulation allows the user to model large complex systems and hence is a popular choice for system level modeling. Selecting a proper language is probably the most important step in the process of developing a simulation model. An incorrect decision during this step may lead to long development times, incomplete studies, and failures. There are four choices : *a simulation language, a general-purpose language, extension of a general-purpose language, and a simulation package.*

Simulation languages such as SIMULA[33] and SIMSCRIPT[191] have built-in facilities for time advancing, event scheduling, entity manipulation, random variate generation, statistical data collection, and report generation. These languages allow the analyst to spend more time on issues specific to the system being modeled rather than worry about issues that are general to all simulations.

A general-purpose language such as C or FORTRAN is chosen for simulation purposes primarily because of the analyst's familiarity with the language. It may also be that deadline requirements do not allow time for him or her to learn a new simulation language.

An extension of a general-purpose language such as GASP (for FORTRAN) is another alternative. These extensions consist of a collection of routines to handle tasks that are commonly required in simulations with the aim of providing a compromise in terms of efficiency, flexibility, and portability.

Simulation packages such as QNET4 and RESQ[127] allow the user to define a model using a dialog. The packages have a library of data structures, routines, and algorithms.

Simulation languages can be classified into two categories, continuous simulation languages and discrete-event simulation languages, based on the types of events they simulate. Continuous simulation languages are designed to handle continuous-event models that are described by differential equations. Discrete-event simulation languages such as SIMULA, GPSS, SIMSCRIPT and GASP are designed to handle discrete-state changes.

An Overview of Some Simulation Languages :

This section provides the reader with a brief summary of some of the popular simulation languages and tools being used in industry and academia today. Simulation languages can fall into one of the two broad categories : flow-oriented languages, and statement-oriented

languages. Statement-oriented simulation languages closely resemble general purpose programming languages such as C or FORTRAN. Flow-oriented languages provide flowchart-like symbols which can be used to construct graphs representing system behavior. There also exists toolkits that serve to extend the original language with simulation capabilities.

SMPL[125] is a general purpose discrete event simulation library written in C. SMPL is portable and uses an event oriented approach.

YACSIM[105] is a process-oriented discrete-event simulator implemented as an extension of the C programming language.

SimPack[75] is a collection of C and C++ libraries and executable programs for computer simulation. Different simulation algorithms are supported including discrete-event simulation, continuous simulation and multi-model (combined) simulation. SimPack provides the analyst with a set of basic utilities that can be built upon to construct special purpose simulation languages. SimPack's discrete-event simulation is an event-oriented approach at the basic level. SimPack also provides a basic X Windows based graphical user interface.

SIMULA[33] is a general purpose language in the style of ALGOL. The language supports object oriented features including encapsulation, inheritance, and polymorphism. SIMULATION, a system class of SIMULA is a process oriented language supporting discrete-event simulation.

CSIM[184] is a process-oriented, general purpose simulation toolkit written with C language functions. The toolkit has been used by programmers to create and implement process-oriented, discrete-event simulation models of computer systems, and software systems including applications executing on multiprocessor systems.

SIM++[224] is a general purpose simulation language based upon C++, that permits writing process-oriented discrete-event simulation models. SIM++ is currently available for PC/AT running Zortech C++ 2.0 or later, and for DECstation 3100/5000 running ULTRIX 4.2 and the AT&T CC.

SIMAN[160] features simulation and analysis of discrete (process or event oriented) and continuous systems (algebraic, difference or differential equations). It is a flow oriented language with the system being represented as linear top-down flow-graphs which depict the flow of entities through the system.

SIMSCRIPT II.5[191] is a general purpose process oriented programming language with structured programming constructs. It provides advanced GUI features to analysts including pull-down menus, push buttons, scrolling text windows and dynamic graphs and meters.

MODSIM II[27] is a high level Modula-2 based object-oriented language with multiple inheritance, message passing, dynamic object creation, dynamic method redefinition and separate compilation of modules. The compiler compiles the source code to C. The programming environment includes a symbolic debugger, a *genealogy* browser with a cross-referencer, a file manager, and a compilation manager.

SLAM II[153] is a simulation language that combines process, event and continuous views on a model. A simulation begins with a *network model* or flow diagram showing the flow of entities. A SLAM II network is made up of nodes at which processing is performed. Common functions are entering and leaving the system, reserving resources, starting and stopping flows etc. Animations can be created by first designing the scene setup and then writing a *script*. Scripts may be written using a forms based system. The script specifies which animation sequence should occur when a simulation event happens.

HOCUS[163] is a simulation package supporting discrete-event simulation modeling using the activity-oriented approach. Activity scanning centers on the definition of activities in a model. Entities are assumed to flow through the model waiting for other entities before engaging in activities in a certain order.

DEMOS[163] is an process-oriented discrete-event simulation implementation on the SIMULA language. DEMOS has a graphical description language and versions exist under MS-DOS and X Windows, both written in SIMULA. A *Demographer* front end as it is called supports hierarchical modeling with sub-processes, based on extended activity diagrams.

2.2 Modeling Tools :

Modeling tools provide the analyst with user friendly interfaces and ease the burden of program development. These tools can be used by the analyst at the cost of flexibility as these tools make assumptions about the type of system

being modeled. Modeling techniques can be analytic, numerical, or simulation based. Analytical techniques provide general models which may be solved symbolically for the steady state measures of that system and can be used to efficiently explore ranges of parameters. Unfortunately, only a very restricted set of models have such solutions. Even fewer have exact solutions, leading to the necessity of finding approximation techniques. Analytical techniques are usually applied to queuing networks, where the structure of the network allows good rules for finding appropriate models, notably those in the class known as BCMP networks.

Somewhere between analytical and simulation models, it is possible to use numerical techniques, where the steady state behavior of a system is found without detailed simulation, but only in terms of a given set of parameters.

The system level modeling tools discussed in this paper into three categories : queuing network based, petri-net based and tools that use other techniques to model a system.

Queuing network based tools :

QNAP[212] uses a high level textual language for the description of models which is then compiled into a form solvable by a range of solvers offered. Solvers include exact solvers for BCMP networks, numerical solvers for less restrictive models, a Markovian solver for reasonable sized models preserving Markovian assumptions and a simulation solver for any model describable in the QNAP language. The QNAP language is structured around entities called *service centers* which could be simple server nodes as in queuing networks or may have more complex behavior, described in an algorithmic language. Options for tracing and debugging simulations are also supported in current versions of QNAP.

QNAP is now a part of Simulog's MODLINE[164] modeling tool incorporating several features developed during the ESPRIT II IMSE project. One of the features is a graphical user interface allowing models to be built from a menu of symbols as a queuing network . Nodes can be parameterized to define a complete model and service center nodes can have QNAP code associated with them]to provide general descriptions of behavior. A second feature is an experiment description facility which allows analysts to describe repeated runs of a model with varying parameters and collects outputs from each in a systematic manner. MODLINE provides features for animation of simulation execution and selective instrumentation of models.

MACOM[114] was developed to support **M**arkovian **A**nalysis of **COM**munication systems. The model view consists of sources, sinks and service and control elements. MACOM allows desired measures and derived statistics and input parameters (including experiment series) to be defined by so called *evaluation descriptions*. A GUI is used to construct the models and evaluation descriptions. MACOM solves models by numerical evaluation of the markovian chain. MACOM runs on SUN workstations and its graphical interface is supported under SunView and X Windows.

The Computer-Aided Performance and Reliability Evaluation System (CAPRES)[108] is a queuing network-based tool for evaluating the design of large-scale, parallel, fault-tolerant computer architectures. CAPRES uses the analytic approach for estimating performance, and can apply one of the following techniques : modified mean value analysis; flow-equivalent aggregation via Chandy *et al.'s* theorem[21], or modified linearizer algorithm. CAPRES can predict the performance, reliability, and performability of a system response times, queue lengths, nodal and system throughput, and component utilization.

The Graphical Input Simulation Tool (GIST)[192] is a transaction-oriented, discrete-event simulation tool for developing extended queuing network models. GIST models are passed through a translator which generates source code for a simulation compiler. Performance statistics for the model are collected including queue length, queue waiting time, and number of waiting jobs.

The Performance Analysis Workstation (PAW) [133] is a graphical tool that supports the development of queuing network models. Models are defined in terms of nodes and uni-directional links. PAW provides the analyst with three tools : a graphics editor, a text editor, and a simulator. The graphics editor allows the user to specify the network topology as a diagram. Parameters associated with each node are entered via the text editor through the use of forms. The simulator allows two modes of execution : continuous or step. The simulator also allows model execution to be traced and features the facility to provide periodic snapshots.

The Research Queuing Package (RESQ)[127] is a modeling tool that supports the development and analysis of extended queuing network models. A model consists of nodes, queues, jobs, routing rules, and routing chains and is specified through the use of a RESQ language. Models can be solved either analytically using the Mean Value Analysis algorithm, or through simulation. RESQ can determine resource utilization, throughput, mean queue lengths, mean queue time, queue length distributions, queue time distributions, and statistical analysis of tokens.

Petri Net based tools :

GreatSPN[46] has evolved from a fairly simple tool for graphical construction and numerical solution of GSPNs. Model construction is supported by placing and linking icons from a menu, representing places, transitions and arcs. The resulting net may be analyzed for structural and behavioral properties, such as deadlocks and invariants. The model is also solved by numerical techniques based on generating the underlying Markov chain.

DSPN[122] Express is offered by the Technical University of Berlin. DSPN Express allows numerical solution of models incorporating deterministic time delays in transitions. DSPN Express does not support simulation.

QPN[25] is a Petri net modeling tool from the University of Dortmund. The tool is similar in its general appearance to GreatSPN. It supports *timed places* as well as timed transitions. A timed place corresponds to a service station of a queuing network for which a Petri net equivalent is known. Solution of the underlying Markov chain is performed with Usenum, a package for solution of large Markov chains, also developed at the University of Dortmund.

ADAS [4] is an integrated set of Petri net-based tools that supports the development of hierarchical models. The tool set includes : a graph editor that is used to create and modify *directed graphs*; a Petri net simulator that verifies the correctness of a software directed graph by converting it into a Petri net and simulating it; a Petri Net Analyzer that processes the results of the Petri net simulator and produces performance analysis reports; a high level hardware description language that verifies the correctness of the hardware graph by generating a HDL program and simulating it; and a software functional simulator that supports the development of either C or Ada modules for modeling the software operations associated with a graph node.

Modeler is a Stochastic Timed Attributed Petri Net (STAPN)-based simulation tool that provides a GUI based environment for model development. STAPN is an extension of Petri nets that supports branching, time delays, and inhibitors that can prevent a node from firing even when all required inputs are enabled. Modeler as presented in [35] is a prototype model that has limitations on the number of input and output nodes, a limited statistical output, and limited ability in displaying model characteristics.

The System Architects Apprentice (SARA)[66] is an environment for the analysis of concurrent systems. The tool set provided with SARA includes : a structure language (SL) that provides a set of primitives for defining a model's structure in a nested, hierarchical manner. SL is responsible for managing resources and ensuring that the interface between modules remains consistent; a Module Interface Description (MID) that acts as a support tool for SL which helps establish accessibility of resources; a Graph Model of Behavior (GMB) that provides primitives akin to Petri nets for specifying and analyzing the control and data flow behavior of a system; and a model library that has facilities for storage and retrieval of model components. SARA calculates performance parameters comprising of mean utilization, mean queue size, mean waiting time, queue size distributions, and confidence measures for all modeled resources.

Miscellaneous Modeling Tools :

This section presents some system modeling tools that do not use the classical techniques of modeling. The tools presented here are based on formal language specifications, performance process algebras and other recent modeling techniques.

SimPar[94] is a modeling environment for the performability analysis of massively parallel computer systems. Performability analysis in **SimPar** comprises both performance and dependability analysis by considering the performance degradation in the presence of component failures. **SimPar** uses the process-based simulation engine and the error injection capabilities of the DEPEND tool[176]. **SimPar**

uses a technique called *conjoint simulation*[90] which is based on the partitioning of the system model and on the combination of various modeling techniques. A so-called architecture-workload model (AWM) comprises the architecture and workload of the target system and relies on the object-oriented and process-based paradigms. A failure-repair model (FRM) represents the occurrence of component failures and control of fault-tolerance and maintenance mechanisms. Performability analysis involves conjoint simulation of the AWM and FRM models.

The Parallel Architecture Research and Evaluation Tool (PARET)[150] is an interactive, animated environment for analyzing multicomputer systems. Modeling a system in PARET comprises developing three separate specifications : characterization of the application software, characterization of system functions, characterization of the interconnections. All specifications are modeled as *directed flow graph* objects. PARET supports animation and interactive monitoring of the simulation of the model.

Process algebras have evolved recently to address some of the shortcomings of simple Petri nets for behavioral analysis of computer systems. Formal protocol languages and system description languages that have often been heavily influenced by process algebras are being investigated as a means for providing performance models directly from system descriptions and specifications. Early experiments incorporating such an approach include TIPP[88] from Universitat Erlangen-Nurenberg, and PEPA[84] from the University of Edinburgh. TIPP was an attempt at a performance modeling extension to a process algebra. TIPP is similar in its algebraic notation to Milner's Calculus of Communicating systems (CCS). TIPP has demonstrated that the notation could express models outside those easily dealt with by previous performance modeling formalisms and also the potential for solving such models by numerical techniques. The PEPA (Performance Estimation Process Algebra) is also similar to CCS in its algebraic structure. It adds to the behavioral analysis capabilities of CCS by being able to generate a Markov chain from the state transition model underlying the algebraic description. The PEPA workbench allows models written in PEPA to be entered and their underlying Markov Chain to be generated in a form suitable for processing by a backend written using the *Maple computer algebra package.*

A number of groups in Europe are experimenting with the automatic generation of performance models from formal specifications. LOTOS is the CCITT recommended protocol specification language based on process algebra and combining the features of CCS and Hoare's Communicating Sequential Processes (CSP). QNAP models have been generated from LOTOS specifications as a part of the ongoing ESPRIT project [213].

2.3 Measurement Tools :

The preferred method of evaluating computer systems through the measurement approach is through the use of benchmarks. A benchmark is a set of executable instructions which may be used to compare the relative performance of two or more computer systems. A benchmark is usually composed of computer programs, but may also include scripts of narrative instructions that direct a person or a machine to perform certain specific tasks during the course of the comparison test. The process of benchmarking is conducting controlled experiments to collect measures of system performance which may be compared from one system to another.

Numerous benchmarking suites are available for most commercial computer systems. Some well known benchmarks are described in [102]. An overview of some benchmarking techniques is presented next.

The *sieve* kernel has been used to compare microprocessors, personal computers, and high-level languages. It is based on Eratosthenes' sieve algorithm and is used to find all prime numbers below a given number *n*. The *Ackermann function* has been used to assess the efficiency of the procedure-calling mechanism in ALGOL-like languages. The average execution time per call, the number of instructions executed per call, and the amount of stack space required for each call are used to compare various systems. The **Whetstone** suite exercises such processor features as array addressing, fixed- and floating-point arithmetic, subroutine calls and parameter passing. The LINPACK suite consists of a number of programs that solve dense systems of linear equations. The LINPACK benchmarks are compared based on the execution rate as measured in MFLOPS. The **Dhrystone** kernel

contains a number of procedure calls considered to represent systems programming environments. The benchmark is a measure of integer performance; it does not exercise floating-point or I/O processing. The SPEC (Systems Performance Evaluation Cooperative) bench mark suite stresses primarily the CPU, Floating Point Unit (FPU), and the memory subsystem.

Measurement of a computer system can be performed by hardware or software monitors. Some hardware systems offer facilities that can be used for analyzing performance parameters, like special counters for recording events. The information can then be read by the monitoring software and processed. When dealing with events on a bus or network link, special hardware is required. In micro-coded architecture, monitoring facilities could be provided at that level for event capture. Software monitoring can be provided at many levels - recording very low level activities like disk accesses etc., at intermediate levels such a operating system calls, or at high levels, recording application level activity such as database requests. Mapping requests at one level onto requests at another is a difficult activity at best. In the following paragraphs, We survey some tools that are specifically oriented towards monitoring hardware level performance.

The Test and Measurement Processor (TMP)[220] is a multicomputer monitoring facility that monitors the behavior of a MC68000-based distributed system. TMP consists of a host test station and a set of local monitors, all interconnected via a monitoring network. Each local monitor contains a MC68000, an I/O unit for output to a terminal or printer, a network interface unit, and an event processing unit. The local monitors observe and record the bus traffic of the local processor and produce performance summaries. Summaries, which can include the number of messages transmitted and received, elapsed time, execution times, idle times, etc. can be sent to the host test station to be displayed. The TMP can monitor and produce performance summaries at all levels of the hierarchy (system, process and module).

The Vax 8800[47] Monitor is a hardware monitor that collects data on the 8800 processor's program counter and memory bus status. The Vax 8800 monitor comprises of two modules : a histogram module and a Digital DMF-32 synchronous parallel interface module. The histogram module is responsible for maintaining a count of all machine cycles executed within the 8800 processor. The histogram module can also keep track of stalled cycles and the status of the memory/IO bus at each clock cycle. The DMF-32 module provides an interface to the histogram module for initialization control and downloading of histogram data. The following performance parameters can be determined based on the collected data : opcode execution frequencies, operand specifier frequency distributions, frequency of reads and writes per instruction, frequency of events on the memory/IO bus, read and write hit ratios, and stalled cycles per microinstruction.

Zahlmonitor 4 [61] is a measurement environment for monitoring multiprocessor systems. It includes a set of hardware probe units for the object system components and a PC attached to each probe unit. The PC's are connected to a central control and evaluator station. A global time base is maintained via tightly coupled clocks synchronized by hardware.

Sterling et al. describe a hardware monitor (the Degradation due to Latency and Arbitration (DLA) device) for the CONCERT multiprocessor in [204]. Several sources of performance degradation are identified namely (1) insufficient parallelism in the application, (2) contention for shared resources, (3) overhead imposed by partitioning the problem, and (4) latency of access to objects. DLA was designed to measure the effect of contention and latency at the hardware level. The DLA monitor was capable of accumulating statistics in real time concerning bus utilization, bus requester wait time, memory access latency, and contention for software level semaphores.

REMS (Resource Measurement System) [43] is a tool to aid in the analysis and measurement of hardware performance for shared bus multiprocessors. Events of interest include low level hardware activities such as memory access, cache access, I/O operations, and queueing for shared hardware resources. REMS is composed of a set of sample units connected to an analysis subsystem. The sample units compare the state of a set of signal lines (connected to the system under testing) to a set of patterns. A pattern matching hardware allows for fast comparison of an incoming pattern with a set of patterns of interest. A pattern match may initiate other recording activities including counter sampling etc.

TRAMS[43] (TRAce Measurement System) has been developed at the National

Bureau of Standards. Events of interest are marked by writing to a location in the address of each process. The data written to the address is then recorded by the measurement hardware along with a 32 bit time stamp, the processor number, and the execution mode (user or system) of the processor.

ATUM (Address Tracing Using Microcode)[193] collects traces of addresses issued from every instruction executed by a VAX 8350 multiprocessor. ATUM is composed entirely of microcode that augments the standard 8350 microcode. As each memory request is issued by the processor, ATUM writes a record of the request, including the virtual address and the type of access, to a block of memory reserved for ATUM use. Traces from ATUM experiments have been used in studies of cache performance and to support cache models and other performance models that rely on memory reference patterns.

3 PROCESS LEVEL ANALYSIS :

This section surveys tools and techniques that have been developed specifically to analyze and predict the performance of a computer system at the process level. It should be noted that a number of the system level tools discussed in previous sections can also be used for this purpose. A number of these tools use analytical techniques to obtain a rough estimate of the performance parameters and simulation techniques during advanced stages of the prediction process.

3.1 Modeling Tools and Techniques :

Modeling tools (analytic and simulation based) and techniques have been developed to predict the performance of software systems. There has been extensive work done in performance prediction based on statistical and probability theory methods. Parallel programs can be modeled in terms of distribution functions, random variables, regression models, stochastic processes, markov processes and chains, queuing networks, petri-nets etc. and performance parameters can be obtained. F.Sotz[198] provides an approximation technique to estimate the runtime of a parallel program which is modeled as a stochastic graph. Tasks represent nodes in the graph. The runtime variables of the tasks can be distributed deterministically or exponentially. The technique is based on transient state space analysis. N. Yazici-Pekergin and J.M. Vincent obtain stochastic bounds on execution times of parallel programs assuming the availability of an unlimited number of processors. The execution times of parallel tasks are random variables distributed identically. F. Hartleb and V. Mertsiotakis[92] derive upper and lower bounds for parallel programs as a means to experiment with mapping and implementation alternatives. Parallel programs are modeled as a stochastic graph and the runtime behavior of a specific processor is described by a random variable. Simulation techniques such as emulation, Monte Carlo, trace-driven and discrete-event simulation can be used to evaluate the performance of program models (graph, queuing models, petri-net models etc.). J. Prost and S. Kipnis[166] describe a multi-level trace-driven simulation approach in order to analyze the performance of programs for distributed memory parallel systems. The trace consists of a sequence of events to be simulated. A parameterized model of the target architecture is incorporated and four hierarchical simulation levels allow the user to examine performance parameters at the user, library and communication levels. J. Bruner *et al.* [38] create instrumented profile runs of a parallel program, which serve as input to an event-driven simulator. This approach is aimed at determining the maximum available parallelism in a program. A Computer Architecture Research Language (CARL) is used to model the underlying architecture. H. Mierendorff et al.[136] evaluate the performance of parallel programs on distributed memory multi-processor systems. They introduce an analytical approach considering message routing, algorithm structure and data mapping. The tool developed can model large systems both in terms of architecture and algorithms.

Benchmarking models, long used for performance measurement at the system level is now a popular performance prediction approach to support the optimization effort at the process level. V. Sarkar[180] describes a general framework for determining average program execution times in the PTRAN project by using frequency information and pre-measured execution times of primitive operations. Balasundaram et al.[23] describe a performance estimator to select a data distribution strategy based on runtime information. It is limited to programs utilizing the loosely synchronous communication model. A set of kernels for operations on a single processor, and loosely synchronous collective communication routines

on a parallel architecture are incorporated to train the estimator. A parallel program is parsed for detection of pre-measured kernels. The estimated runtime of this program is derived as the accumulated time of all kernels. N. MacDonald[124] estimates the performance of a subset of Fortran77 programs using analytical time formulae, considering only primitive control flow. Benchmarks are used to pre-measure primitive code kernels and these pre-measured times are used as parameters in the analytical time formulae.

W. Abu-Sufah and A.Y. Kwok[1] present a set of performance prediction tools developed for the Cedar multi-processor system. Their approach involves analytical and simulation techniques incorporating guessing for unknown parameters. Analysts can choose from either of these techniques depending on the accuracy required (analytic for coarse grain and simulations for fine grain).

D. Atapattu and D. Gannon [19] obtain estimated runtimes for parallel FORTRAN programs in order to support program transformation. An analytical model of the bus behavior for the Alliant FX/8 is incorporated assuming exponentially distributed processes and a queuing model. Their estimates are algebraic expressions of unknown loop bounds and number of processors.

Modarch[225] is an environment dedicated to performance evaluation of distributed computing systems. Modarch helps find the best fit between hardware configuration and software applications. Modarch is built upon the Modline environment and uses a dedicated version of the QNAP2 simulation software. Modarch features a graphical programming interface that allows the analyst to specify the software and hardware architecture. The tools included in the Modarch environment are : (1) The *Experimenter* which automatically generates simulation runs of a model. The experimenter runs the model for each possible value of the input parameters and stores output results. (2) The *Analyzer* allows interactive extraction of output data and visualization as graphs : lines, bars, pie charts etc. (3) The *Reporter* generates and compiles all the information relevant to the report subject within the study.

AIMS (Automated Instrumentation and Monitoring System)[226] is an ongoing effort at NASA. AIMS consists of a suite of software tools for measurement and analysis of performance. Our area of interest is the modeling facility provided with the AIMS environment. The Modeling Kernel (MK) is a facility in AIMS for modeling parallel programs. MK supports simulation-based and analytical approaches to performance prediction and scalability analysis, automates the process of building and simulating parallel-program models. Based on such models, users can obtain asymptotic performance characteristics for either the entire program (process level) or individual components (module level). The main component of MK is GPPM (Generator of Parallel-Program Models). GPMM models parallel programs at the coarsest level, capturing only the duration of sequential blocks, the lengths and destinations of messages, loop bounds and conditional branch probabilities. All references to I/O and memory are ignored. The model structure mirrors program structure and is derived from parse trees, one per FORTRAN or C module.

Axe[221] is an integrated set of tools for the analysis of algorithms and partitioning strategies on mesh-connected concurrent processors. The tool set includes a compiler/translator, a simulator, a monitor and an *experimentation executive* for processing user generated commands. The user specifies the program to be analyzed using a behavior description language. The behavior description language allows the user to specify the model using similar constructs as the application implementation language, except that the execution time of the statements is simulated. In addition to providing a program description, the user has the ability to define characteristics of the run-time environment, including message I/O overhead, process creation overhead, communication link bandwidths, number of nodes and the amount of memory per node. The user can also select from a limited set of built-in topologies, routing algorithms, partitioning algorithms, and scheduling algorithms. Performance data is generated by discrete-event simulation of the model.

The Network Emulation Tool (NET)[20] is a computer network simulation that supports the analysis of distributed operating systems and distributed databases. NET provides a default network description which can be altered by the user to represent a limited set of networks. Several network parameters can be user defined including message delay, message loss, message duplication, node failure rate, and network partitioning. An user defined network description can be created when the default

description is inadequate. NET generates statistical output on network performance as well as algorithm performance.

The Rice Parallel Processing testbed [54] is an execution-driven environment for the analysis of concurrent programs. Actual workloads are executed on the target computer to obtain realistic processing delays while all interprocess communications and interactions are simulated allowing for a variety of architectures to be evaluated. The user must provide three types of input : a concurrent program, a simulation model of the architecture, and a process-to-hardware mapping. The environment comprises of (1) *Concurrent C* -This version of C supports parallel programming. The program to be evaluated as to be written in this language. (2) *C Simulation Package* - This package is a discrete-event simulator for event queue manipulation, data collection, and tracing. (3) *Architecture Simulation Preprocessor* - The preprocessor inserts simulation primitives into a Concurrent C program. These primitives represent inter-process communication and synchronization delays. (4) *Timing Profiler* - The profiler is an assembly language analyzer that estimates execution time of sequential code segments. (5) *Simulation Tool Interface* - The user interface is menu driven and supports windowing. (6) *Parallel Tracer/Debugger* - This tool supports a windows-oriented user interface for monitoring and controlling model execution. (7) *Library* - A library is provided for storing concurrent C programs and architecture models.

QASE[227] is an analytic and simulation modeling tool for distributed client/server applications. QASE's system description is a hierarchical entity-attribute specification. Entities in a system include execution flow diagrams, workloads (periodic or random), hardware diagrams (processor, storage and communication architectures), software, data (data stores and flows), operating systems, communication protocols, and allocations. QASE uses multiple evaluation techniques using the analytic approach to evaluate feasibility of alternate system descriptions and discrete-event simulation for detailed analysis during final stages of design. QASE also supports automatic model generation by populating the model with performance metric data collected using HP's MeasureWare Agent.

The Vienna FORTRAN Compiler System (VFCS)[45] has a parameter based performance prediction tool in its tool kit. Parameter based performance prediction of FORTRAN programs is made possible by this tool. Workload parameters including work distribution, number of data transfers, transfer times, network contention, cache miss ratio, and main memory performance can be modeled analytically. The parameters are modeled and expressions are derived for statements, loops, procedures and the entire program. The flow variables (control and data) are not guessed (specified by the designer) but are estimated through profile runs of the code. Current work is focused on training the tool by running different program profiles under different workload conditions.

PEPP (Performance Evaluation of Parallel Programs)[59] is a modeling tool for creating and evaluating stochastic graph models of parallel and distributed programs. PEPP offers functions for graphical model creation and various evaluation methods for calculating the mean runtime of a program. PEPP supports the idea of *model-driven monitoring*, where modeling and monitoring are integrated into a framework to support easier evaluation, tuning and debugging of parallel and distributed systems. PEPP is implemented in C with the graphical user interface implemented on top of the X Windows system.

MENTOR (Model based EveNT Trace analysis suppORt system)[60] is an expert system which assists in the trace evaluation of parallel and distributed programs by incorporating knowledge about the program under investigation into a trace analysis environment SIMPLE[107]. The knowledge is derived from stochastic graph models created with PEPP.

3.2 Measurement Tools :

Process level measurement tools require instrumentation of the relevant code (process code or operating system code) for which performance data is to be obtained. One of the major concerns facing designers of such tools is the perturbation introduced in the performance data obtained as a result of the measurement process. Instrumentation of the code has to be as non-intrusive as possible so as to obtain accurate results. The following sections survey some measurement tools designed specifically for the process level.

Process Level Measurement Tools

The Berkeley UNIX Monitor[137] is a software monitor within the kernel for measuring the performance of a distributed program. The monitor is a distributed program capable of executing its functions on a user specified processor. The monitor provide four functions :

- Meter - detects and records events within the kernel so as to produce a trace. Trace data can include creation/destruction of processes, starting/stopping of processes, and inter-process communication.
- Filter - extracts user specified trace information from the trace data generated.
- Control - provides an interface to the user to control the measurement process.
- Analysis - analysis routines can be defined by the user to summarize and report on the filtered traces.

The UNIX gprof utility[89] introduces the concept of a dynamic call graph generation for an execution of a program. The dynamic call graph contains one node for each routine that is invoked as the program executes. Each directed arc in the graph connects a caller with a callee. The gprof routines build the dynamic call graph from a program run. At compile time, calls to an event recording routine are inserted at the entry to each subroutine. When the subroutine is called, an arc between the caller and the callee is recorded in a table. The graph is generated by a post processor after termination of the process.

Monit[111] is a performance monitor for the Sequent Balance 8000 system. Events of interest included task and process creation and termination, entry to and exit from resource queues, and a general "value trace" event. Active recorders log event occurrences to a buffer in memory. A separate process is responsible for transferring the buffer contents to permanent storage.

Radar[119] is a debugging tool to assist in analysis of distributed applications. The applications execute on a network of PERQ workstations. The events of interest for the developers of Radar included process creation and termination, message transmission and reception, port creation and a general purpose event. Events are recorded by the node in which they occur. Each event is marked with an event number as there is no concept of a global synchronized time. The event recorder copies the content of each message as a part of the event record. This feature facilitates the replay of the entire experiment in "single step" mode.

PCA (Performance and Coverage Analyzer)[62] is a performance measurement tool designed for the VAX architecture. PCA has been used for both uniprocessor and multiprocessor applications. Measurement experiments are divided into two phases: the collection phase and the analysis phase. The collection phase involves sampling the program counter at intervals determined by the system timer (ten milliseconds). Histograms of program activity by subroutine, or even by line of source code can be generated. The time cost of work done by the low level subroutines can be propagated back to statements within the higher level subroutines when the call stack information is accumulated. PCA allows the insertion of software trace markers that allow other statistics namely (1) number of invocations of each selected subroutine or code fragment, (2) the number of page faults incurred by each module, routine or line of code, (3) frequency of requests for system services by location in the application.

The FORTRAN Analyzer[123] is a syntax driven software that inserts monitoring code into an American National Standard FORTRAN program. Parameters passed to the monitoring routine include the segment being monitored and the monitoring routine's entry point. A code segment is enclosed between the entry and exit points. Thus this tool can be used to monitor performance at the module level by appropriate instrumentation. The storage requirements for instrumented programs may increase by 26 to 55%.

Parasight[18] is an environment for performance analysis of sequential and parallel programs. The platform for Parasight is UNIX on the Encore Multimax which is a shared-memory multiprocessor system. Parasight is executed concurrently with the program to be monitored which is embedded within the Parasight environment. The environment, upon startup initializes a multitasking environment. The monitored program is loaded into this environment and a memory resident symbol table is created. The code is executed concurrently with Parasight programs that monitor shared memory. Parasight routines can be offloaded to processors other than the one being used by the monitored code to reduce interference. Parasight provides breakpoints that can be created and deleted dynamically at run time.

The Parallel Software Environment (PSE)[228] is a performance analysis product from DEC that includes a loop-capable and parallel profiler for high performance FORTRAN. The profiler provides information about the time spent in logical sections of the code such as do-loops. The profiler allows programmers to view program-unit and statement-level timing information about parallel execution. The performance information also includes communication times included with individual FORTRAN statements.

The Programming and Instrumentation Environment (PIE)[120] is a framework for developing techniques to predict, detect and avoid performance degradation in parallel and distributed programs in a shared memory multiprocessor environment. PIE supports the analysis of parallel process composition, communications, and data partitioning. PIE is implemented on top of the *Mach* kernel. PIE provides a customized visual editing system through which the user identifies the principal programming constructs. PIE provides a meta-language to support the development of parallel algorithms for observation and analysis. The meta-language is used in conjunction with Pascal and extends its capabilities by providing parallel functionality such as synchronization, access to shared data, etc. After the source code visualization, PIE allows for automatic observation of constructs within the code. PIE's instrumentation is currently done using software instrumentation techniques. An *Implementation Assistant* tool provides semantic support for parallel program development. The tool helps predict program performance before implementation and assists the user in selecting a parallel implementation. PIE's visualization utilities include histograms and time-lines.

The JADE[220] programming system is a distributed monitoring facility consisting of two parts : data detection and collection, done by so-called channel processes, and data analysis and presentation, done by so-called consoles. JADE extends debugging support to distributed applications based on inter-process communication.

The INCAS[220] project at the University of Kaiserslautern has developed a tool for measuring the performance and observing the behavior of distributed systems during execution. A hardware support module, called Test and Measurement Processor (TMP) is integrated into each node of a distributed system. All TMP's are connected to a central monitoring station via a measurement LAN. Sensor code in the monitored system is reduced to single store instructions for event signaling, leading to very low interference.

Sun Microsystems provides SPARCworks[229], a tool to support dynamic analysis and control of multi-threaded programs. SPARCworks supports analysis of the code for potential synchronization errors such as deadlocks and data race conditions. Detailed thread level profiling is also supported.

JEWEL[117] is another distributed measurement environment that consists of four functional blocks :

- the system under test (SUT),
- the data collection and reduction system (DCRS),
- the graphical presentation system (GPS),
- and the experiment control system (ECS).

Measurement data is extracted from the SUT, collected and filtered by the DCRS, and then passed to the GPS for visualization to the experimenter concurrent with the operation of the SUT. Interpreting the visualized data may result in actions e.g. customizing the graphical appearance or taking a snapshot, control requests issued to the ECS , e.g. to change the level of detail, to stop the current experiment, or to set up a new configuration.

SPY[214] is a software monitor that does periodic location counter sampling to determine the performance of an application program. Function calls provided to the user include a setup call that initializes SPY, an activate monitor call that turns on monitoring, and a terminate monitor call which turns off monitoring. The startup call requires that the user specify a histogram array name, array address, and sampling interval. All measurement data is stored within the histogram array in the address space of the program.

TX-2[148] is a time-shared system that provides a hardware monitor for measuring program performance. The monitor has access to the program counter and index registers. The monitor can track events as they occur in the processor and update the relevant parameters. Thus a histogram of the desired parameter is available upon program termination.

MemSpy[132] is a tool that helps programmers identify memory bottlenecks in parallel and sequential programs. MemSpy provides information such as cache miss rates, causes of cache misses, and in multi-processor

systems, information on cache invalidations and local versus remote memory misses.

MTOOL[86] is a tool aimed at detecting regions of a program where the memory hierarchy is performing badly. MTOOL identifies memory bottlenecks by comparing the measured execution time with the predicted time for a perfect memory hierarchy. MTOOL is aimed at FORTRAN programs running on MIPS based workstations.

IPS-2[138] defines a computational hierarchy on the program being monitored. The program is represented as a black box at the highest level. The next level is the machine level where the program is split into several concurrent processes executing on different processors. The third level represents the program as a collection of communicating processes. The final level is the primitive activity level. IPS-2 uses instrumentation probes to generate trace data and then evaluates the performance data. The instrumentation provided includes a *gprof* style profiler that records procedure entry and exit events, and modified run-time libraries.

The *Annai*[48] tool Environment is intended for the development and performance evaluation of parallel and distributed applications. Tool components include : (1) A Parallelization Support Tool (PST) for data-parallel program development with particular focus on unstructured computations. (2) A Parallel Debugging Tool (PDT) supporting interactive, source-level debugging and global program views. (3). Performance Monitor and Analyzer (PMA) for directed interactive identification and tuning of performance problems. (4) A Common graphical user interface (UI) and tool/machine interface (TSA). We concentrate on the measurement section of the environment, the PMA. Measurement and monitoring of the code is done by instrumenting the communication library and the compilation system. The tool also features dynamic instrumentation and insertion within executables. A run-time execution profile accumulation and event trace buffering is made available to the analyst.

The AIMS suite discussed earlier provides tools to measure the execution of parallel code. AIMS provides (1) *xinstrument*, a source code instrumentor that supports Fortran77 and C message-passing programs written under two communication libraries : MPI and PVM. (2) *monitor*, a library of timestamping and trace-collection routines that run on the IBM SP-2, as well as networks of workstations (including Convex/HP clusters, SparcStations and SGIs). (3) *pc*, a utility for removing the monitoring overhead and its effects on the trace generated.

WAT (Workload Analyzer Tool)[164] is an effort by the University of Pavia in collaboration with the University of Milan. WAT provides cluster analysis and other statistical analysis and is driven by a graphical user interface. It accepts traces in a number of standard formats and further formats can be added by modifying the input section.

The MEasurements Description Evaluation and Analysis tool (MEDEA)[134], supports the analysis of trace data. The various stages of trace analysis include (1) preliminary analysis of trace data to correlate the events recorded during the execution of an application to prepare the data for further analysis. (2) definition of a format which is a subset of performance parameters associated with the current workload component. (3) cluster analysis to allow the identification of classes of events with respect to certain parameters. (4) A fitting module allows compact analytic descriptions of a workload, which represent the variation of workload parameters with respect to independent variables, such as time. (5) A functional description module allows a logical, rather than a physical description of the workload. The workload is viewed in terms of membership of components to a specific cluster, rather than in terms of overall resource utilization such as processing time and (6) data visualization allowing interactive examination of the workload models.

SP[140] uses hierarchical structuring of systems into components and modules, allowing workloads at different levels to be mapped onto each other. A so called *complexity function* is defined as how much work, in terms of memory, communication capacities and processor usage at one level corresponds to units of work at another. SP can be used for mapping measurements onto required input parameters of performance models and for certain simple direct modeling, such as capacity management decisions.

Measuring Operating system performance :
Modern operating systems (Solaris, NT, 95, OS/2) provide on-line performance meters to provide the user with a continuous visualization of system performance.

Imbench[229] is a suite of portable benchmarks that compares the performance of different UNIX systems. Imbench runs a set of benchmark programs on the target machine in order to obtain performance data. Benchmark results are available for most major vendors (SUN, HP, IBM, DEC, SGI and PCs). Imbench is a free software covered by the GNU general public license. Imbench provides bandwidth benchmarks including cached file read, memory read/write/copy, and pipes. Latency benchmarks include context switching, file system creates and deletes, process creation, system call overhead and memory read latency.

SymbEL (SE)[229] is an interpreted language that acts as a toolkit for building performance tools and utilities. SE provides scripts that build on the basic tools (vmstat, iostat, sar etc.) to provide rule-based performance monitors and viewers. The package includes a Motif based GUI library and a rules library.

The AXXiON[230] performance manager provides performance monitoring for UNIX and Windows NT systems. The AXXiON performance manager can be configured to collect data on real-time performance elements including memory utilization, disk I/O, individual processes and other system/network activities. It delivers snapshots of resource activity correlating performance data from a variety of resources.

3.3 Visualization Tools :

Visualization tools provide the developer with a visual display of program execution. These tools are useful when the behavior of a program cannot be inferred easily by statistical analysis alone. Though visualization tools fall into one of the three categories (measurement, simulation, modeling), they deserve special attention because of their unique graphics features. Visualization tools can be on-line or postmortem tools. Visualization tools that support postmortem analysis do not instrument the code being monitored. They need a trace file as input to be processed and visualized.

CHIRON[87] is a visualization system developed at the University of Cape Town for displaying performance related behavior of shared memory microprocessor applications. The tool is primarily used as a performance debugging tool which can be utilized by the designer to fine-tune or remove performance bottlenecks. CHIRON uses 3D graphics to generate various performance related views which can be scaled, rotated, translated, animated or level-of-detail toggled. CHIRON is used to system performance (emphasis on cache performance), synchronization costs, and data partitioning in a parallel program. It has also been used to optimize sequential programs that waste time through ineffective use of the memory hierarchy.

ParaGraph[218] takes as input trace data generated by the Portable Instrumented Communication Library (PICL), developed at Oak Ridge National Labs and provides visualization of program behavior. PICL can also provide execution trace data during an actual run of a parallel program and the resulting trace data can provide dynamic snapshots of the behavior. ParaGraph organizes the information into various views in an attempt to cope with the massive amount of raw information generated. ParaGraph runs on the X Window System and is implemented using the Xlib library for portability reasons. ParaGraph is designed to be responsive to user interactions while displaying program behavior dynamically. The execution behavior of ParaGraph can be static (initial selection of parameter values) or dynamic (pause, resume, single step etc.). ParaGraph is extensible with users having the ability to add new displays of their own design. This feature supports the use of application-specific displays that can be used to augment the insight that the generic views provide.

PARvis[144] is a tool used on a post-mortem basis to translate a given trace file into a variety of graphical system views which provide a reasonable basis for system understanding and program optimization. PARvis takes as input an *ipd*-generated trace file and extracts graphical information. Different views of the PARvis system include single time system snapshots, animation, statistics and a time-line system view. PARvis is implemented in C and uses the Motif libraries for its graphic capabilities. Hardware platforms include IBM RS/6000, SUN, DEC MIPS and Alpha systems. Extensions to PARvis include display of network activities and flow of messages on different topologies. PARvis provides configuration files that the user can edit from run to run. Parameters include color, layout, fonts etc.

PV (Program Visualizer)[231] developed at IBM, provides continuous visual

displays of the behavior of a program and an underlying system. PV is designed as a tool for debugging and performance tuning and analysis. PV has been targeted to run on shared-memory parallel systems and superscalar uniprocessor workstations (RISC System/6000 with AIX). PV shows hardware-level performance information, operating system level activity, communication library level activity, language run time activity and application level activity. Thus PV can be used as a process level and system level monitor. Users can add their custom configured modules to analyze application specific characteristics. PV has been used to gain more insight into the structure and dynamics of large object-oriented applications, frameworks and libraries.

Pablo[5] is an ongoing research project being developed at the University of Illinois. Pablo is designed to provide performance data capture, analysis and presentation across scalable parallel systems. Pablo is best described as a toolkit for the construction of performance analysis environments. Pablo consists of a portable source code instrumentation subsystem and a performance data analysis subsystem with a trace data meta-format coupling the two. The performance analysis component of Pablo consists of a set of data transformation modules that can be interconnected to form a data analysis graph. Performance data flows through the graph nodes and is transformed to yield the desired performance metrics. Interesting features of Pablo include immersive virtual reality to display performance data and sonification by which performance data is displayed by the use of sonic data presentation.

PARADE (PARallel program Animation Development Environment)[202] is an ongoing project at the Graphics, Visualization and Usability center in Georgia Tech to support the design and implementation of software visualization of parallel and distributed programs. PARADE contains components for monitoring a program's execution, building the software visualization and mapping the execution to the visualization. The primary operation of PARADE is post-mortem visualization with trace files. Software instrumentation is layered with decreasing level of programmer involvement. Instrumentation methods include inclusion of print functions at specific points in the program, overriding the standard communication library with macros and actual modification to the library code to turn on/off trace flags. PARADE visualizations include processor grid view, data distribution views, communication history, message passing views etc. Visualization in PARADE is built around the Polka animation system. Polka provides an object oriented interface to developers which makes coding complicated graphics easier.

4 MODULE LEVEL ANALYSIS :

The tools developed for analysis at this level of granularity rely heavily on mathematical methods to derive time cost equations. Since the system under analysis is not too complex, pure analytical techniques can be used to derive time cost or other performance equations. Many of the tools and techniques described in the section on process level modeling tools can be utilized here. This section surveys tools that have been developed to provide a pure analytic solution to the performance prediction problem.

Metric[217] is an analytic tool for estimating the execution time of simple LISP programs. The user must supply as input (1) a LISP program, (2) a cost table defining the time cost of basic LISP operations, and (3) procedure definitions. The procedure definitions are the previously analyzed procedures of the LISP program and their input is optional. Metric will not re-evaluate these procedures in the event that they are supplied. The time cost of the program is evaluated in three phases. (1) program expressions are converted to cost expressions based on the cost table. (2) Recursive procedure calls are converted into a set of difference equations which are solved in (3) to produce closed form expressions. Metric produces closed-form expressions characterizing the execution behavior of the LISP program and procedure definitions to be used in future analysis efforts.

The Time Cost Analysis System (TCAS)[188] is a Computational Structure Model (CSM)-based tool for analyzing the execution times of parallel computations. The CSM methodology represents a computation as a control graph and data graph. The control graph shows the order in which the operations are performed and comprises of activity nodes (start, operation, decision etc.) and edges. An activation signal propagates through the graph representing an execution thread. A weight associated with the edge specifies the number of times that path should be executed. The data flow graph, similar in structure to the control flow graph, depicts the relationship between the data and the operations of a computation. The computation to be evaluated is written in a Pascal like language,

checked for correctness of syntax and stored in a library for retrieval for future analysis. The computation structure with the flow values (designer specified) can be solved analytically to obtain a time cost expression which can then be used to compute different performance estimates including minimum, maximum and average execution times and to plot time cost curves.

TCAS makes a number of assumptions about the computational environment at runtime. The environment is characterized by a limited number of homogeneous processors that communicate through shared memory and balance the load equally. The last assumption is strengthened by the development of the Optimal Allocation System (OPAS)[168] that provides four allocation policies : (1) Equal, (2) Enough, (3) Sequential and (4) degree of parallelism. OPAS is constructed and operates in a manner similar to TCAS except that time costs can only be solved analytically. OPAS determines an allocation policy leading to minimal execution times and all performance calculations are based on this policy.

The Data Flow Analysis System (DFAS)[169] estimates the execution times of data flow programs. DFAS provides a similar interface as that of TCAS, but the underlying methodology used in the computation of time costs vary. DFAS is based on a token model in which the computation is modeled as a graph. Data flow is modeled as tokens that traverse the graph. A node is activated when the appropriate number of tokens become available on the node's input edges. In addition to the computation, the user should provide (1) the time cost of each node, (2) the time cost of each edge and (3) independent data flows in the computation. DFAS computes minimum, maximum, and average time costs, time cost variance and time cost distribution of the computation.

5 STATISTICAL SUPPORT TOOLS :

Performance analysis of computer systems can produce an abundance of raw data that has to be managed and statistically processed. This has become a serious concern to designers of performance analysis tools for parallel and distributed systems due to the overwhelming amount of data generated. Most of the tools surveyed earlier have capabilities to manage and analyze the generated data or provide users with the utilities to do so. Additional statistical support tools may be needed to augment the existing statistical capabilities of the tool. Table 1 provides a summary of some of the statistical support tools.

6 A CLASSIFICATION METHODOLOGY :

The sheer number of tools available to a performance analyst makes selecting an appropriate tool a challenging task. This section presents a classification scheme that partitions the tools based on their properties. A database can be designed around this classification scheme that would then enable performance analysts to retrieve tool information based on a keyword search.

The classification scheme is presented as a list of figures. The scheme is tree based with each node of the tree representing a property unique to a set of tools. The tool list is refined as we traverse the depth of the tree. Some of the performance tools surveyed in this paper can be placed in more than one category and thus can be placed under multiple nodes in the classification tree.

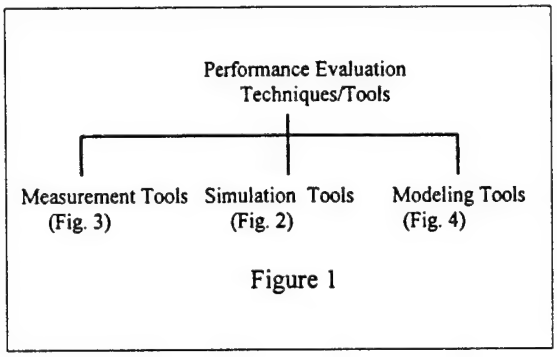


Figure 1

7 CONCLUSION :

This paper has surveyed a number of performance tools that have been or are in the process of being developed in academia and industry. Many of the tools surveyed use sophisticated techniques to simulate or measure the performance of the target system (software and hardware). The tools surveyed fall into three broad categories namely system level, process level and module level. The measurement tools that were surveyed in this paper employ sophisticated techniques to capture system information in an uni-processor/multi-processor environment. A majority of the modeling tools surveyed provide the analyst the option to solve the system model analytically, in addition to detailed simulation capabilities. Thus the performance analyst can use the analytical solution to obtain coarse estimates of system

performance in addition to simulating the model to obtain more accurate estimates at the cost of more computational power. The paper also discussed a classification scheme to aid performance analysts obtain information about the various tools. The information can aid the designer in making a decision as to the type of tool to be used to estimate/evaluate the performance of the underlying system.

TABLE 1
Summary of Statistical Tools
A: basic statistics (mean, median, standard deviation, etc.); B: analysis of variants; C: multivariate analysis; D: regression analysis; E: cluster analysis; F: time-series analysis; G: correlation; H: non-parametric statistics; J: random number generation.

Tool	Description	Platform	Capabilities	Ref. No.
BASS	BASS provides a limited collection of statistical routines.	IBM-PC	A, B, D, F, H	234
BLSS	BLSS is an interactive statistics package supporting matrix operations.	UNIX workstations	A, B, C, D, J	233
BMDP	BMDP provides a comprehensive collection of statistical routines in addition to a database management facility, a full screen editor and graphic facilities.	Various	A, B, C, D, E, F, G, H, I	233, 234
CLAM	CLAM is an interactive environment for matrix-based computations, eigen values, eigen vectors, fast-Fourier transforms, etc.	Various	A, B, C	233
CLASP	CLASP is a tool tailored for cluster and multivariate analysis.	SUN	C, E	233
CSS	CSS is a menu-driven facility providing an extensive collection of statistical routines in addition to a database management facility and a spreadsheet-like editor.	IBM-PC	A, B, C, D, E, F, G, H	234
GLIM	GLIM is an interactive statistics package that provides a facility for interfacing with user-supplied FORTRAN subroutines.	SUN	B, D, G	233
IMSL Libraries	The IMSL library is a collection of over 800 FORTRAN subroutines to support statistical analysis and other areas in applied mathematics such as eigen system analysis, linear systems, differential equations, matrix/vector operations etc.	Various	B, D, G	233, 235
MathStation	MathStation is a general-purpose, interactive tool that supports statistical analysis. MathStation can interface to FORTRAN subroutines and libraries.	SUN	A, B, D, G	233
MATLAB	MATLAB, though oriented for matrix-based computations provides a limited statistics capability. MATLAB supports matrix operations, eigen values, eigen vectors, fast-Fourier transforms, spectral analysis, convolution etc.	Various	C, D	233, 234
Minitab Statistical Software	Minitab provides limited collection of statistical routines.	SUN	A, B, D, F, H	233
Maximum Likelihood Program	MLP is a tool for fitting probability distributions to observed data.	SUN	D	233
NAG FORTRAN Library	The Library contains over 700 routines for statistics as well as other mathematical areas such as linear algebra, differential equations, fast Fourier transforms, interpolation etc.	Various	A, B, D, F, G, H, J	233

Table [1] (continued).

Tool	Description	Platform	Capabilities	Ref. No.
NCSS	NCSS provides a collection of statistical routines and an advanced graphics utility.	IBM-PC	A, B, C, D, E, G, H, I.	234
Prodas	Prodas provides a collection of statistical routines in addition to database and graphics utilities. Prodas can be run either interactively or in batch mode.	IBM-PC	A, B, C, D, E, G, H, I	234
P-Stat	P-Stat provides an extensive collection of statistical routines, a data management facility, a report writer, and a command-generator utility.	Various	A, B, C, D, E, F, G,	233, 234.
RS/1	RS/1 provides a limited collection of statistical routines and supports a graphics capability.	Various	A, B, D, H	234
Sandie	Sandie, originally developed for use in an educational environment, provides a limited collection of statistical routines and supports a multi-window user interfaces.	IBM-PC	A, B, D, G, J	233, 234
SAS	SAS provides a comprehensive set of statistical routines and advanced graphic capabilities.	Various	A, B, C, D, E, F, G, H, I.	234
Sigstat	Sigstat provides a comprehensive set of statistical and graphical routines.	IBM-PC	A, B, C, D, E, F, G, H., I	234
SORITEC	SORITEC provides a limited statistical capability and supports mathematical functions including matrix algebra and analytical differentiation. SORITEC can be executed interactively or in batch mode.	SUN	D, G, H	233
Speakeasy	Speakeasy supports statistical correlation and regression analysis and provides other mathematical functions for solving matrix algebra, set algebra, linear algebra, and differential equations.	SUN	D, G.	233
S-PLUS	The S-PLUS package provides a variety of statistical routines and graphics facilities.	SUN	B, C, D, F, G, H	233
Unifit	Software tool for fitting probability distributions to observed data.	Various	A	235

FIGURE 2 (SIMULATION TOOLS)

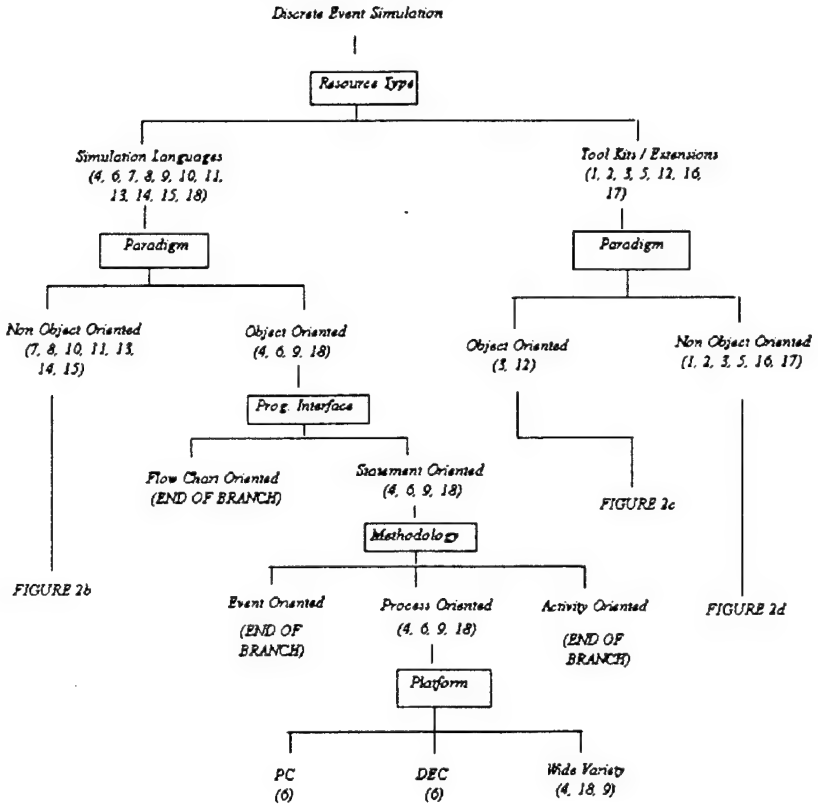


FIGURE 2b (SIMULATION TOOLS Contd.)

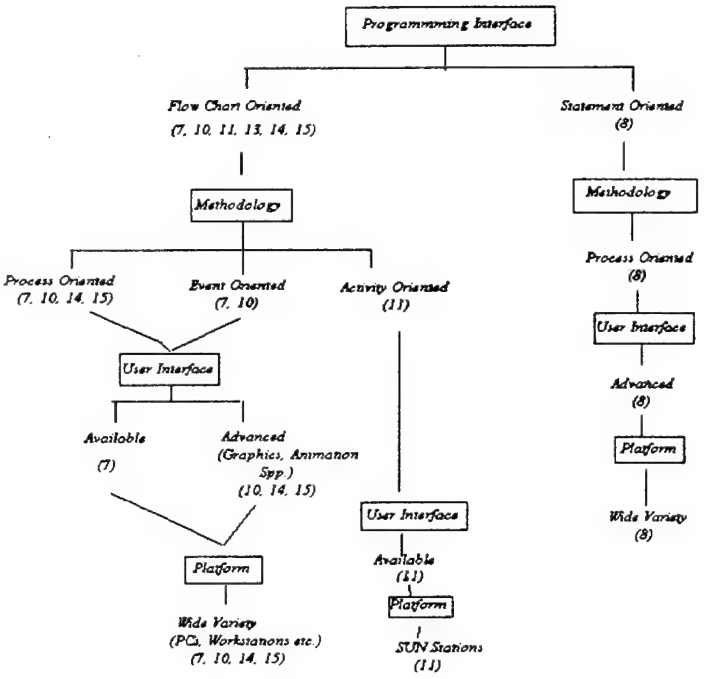


FIGURE 2c (SIMULATION TOOLS Contd.)

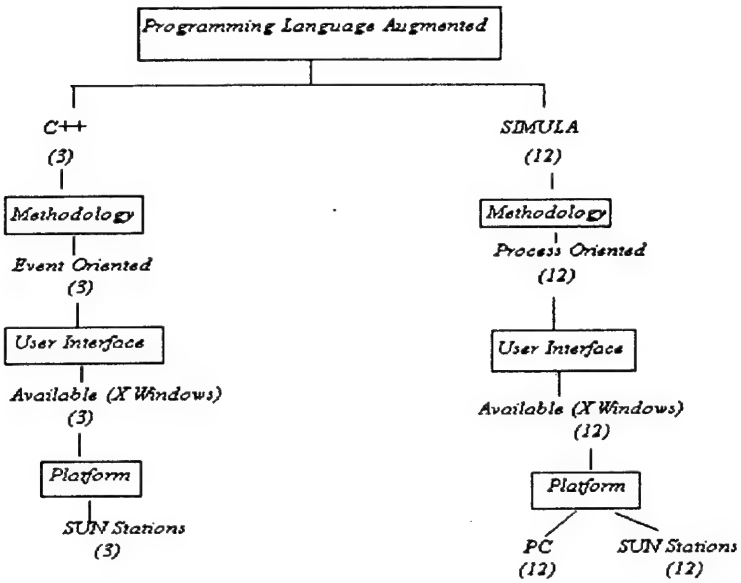
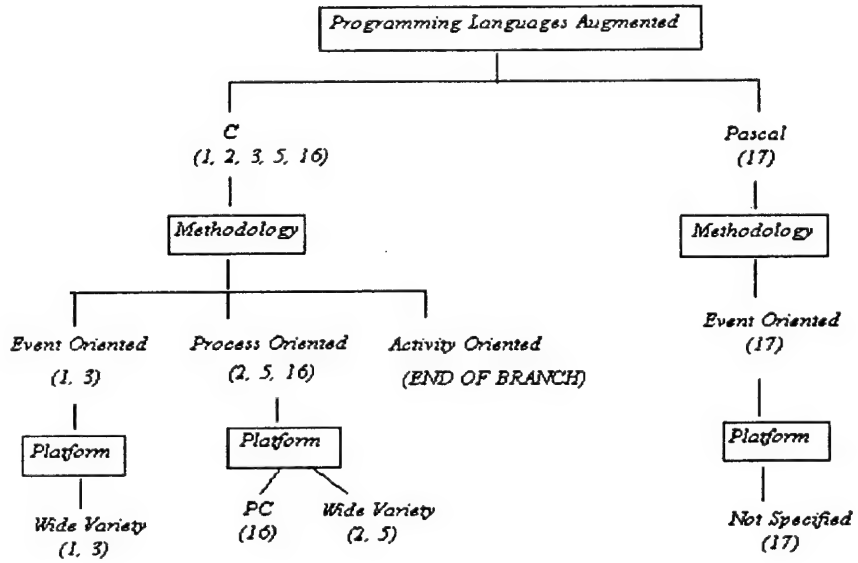
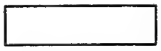


FIGURE 2d (SIMULATION TOOLS Contd.)



Key :



represents a Decision Box.

Simulation Tools List :

(1) SMPL [125], (2) YACSIM [105], (3) SimPack [75], (4) SIMULA [33], (5) CSIM [184], (6) SIM++ [224],
(2) SIMAN [160], (8) SIMSCRIPT [191], (9) MODSIM [27], (10) SLAM II [153], (11) HOCUS [163], (12) DEMOS[163],
(3) FAST [179], (14) GPSS [37], (15) INSIGHT [173], (16) SimCal [129], (17) SIMTOOLS [177], (18) Smalltalk [113].

FIGURE 3 (MEASUREMENT TOOLS)

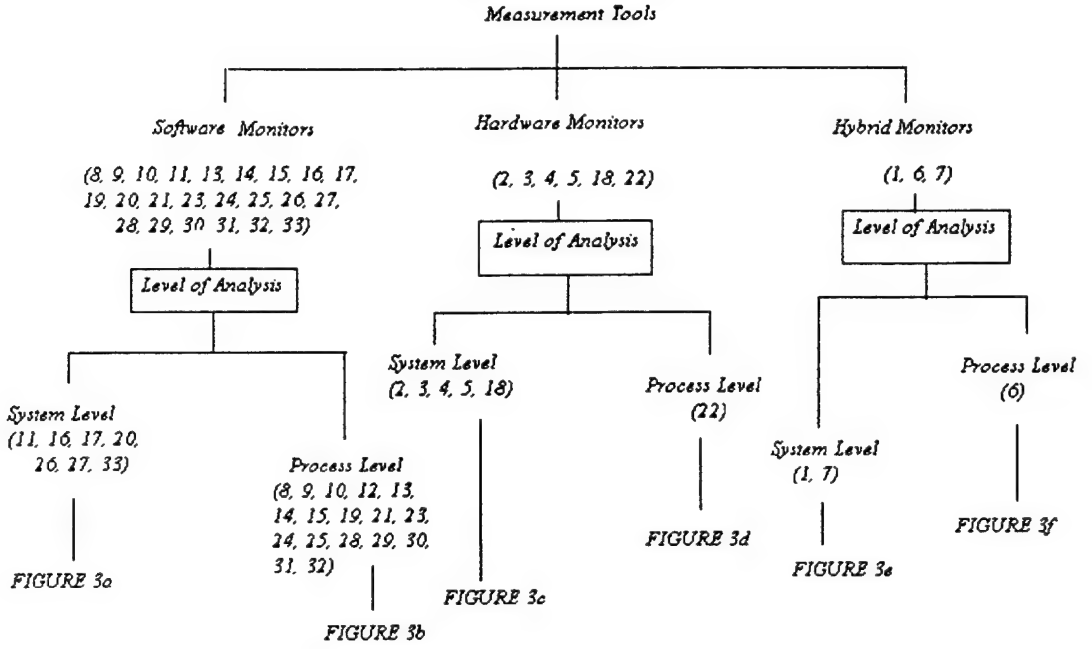


FIGURE 3a (MEASUREMENT TOOLS Contd.)

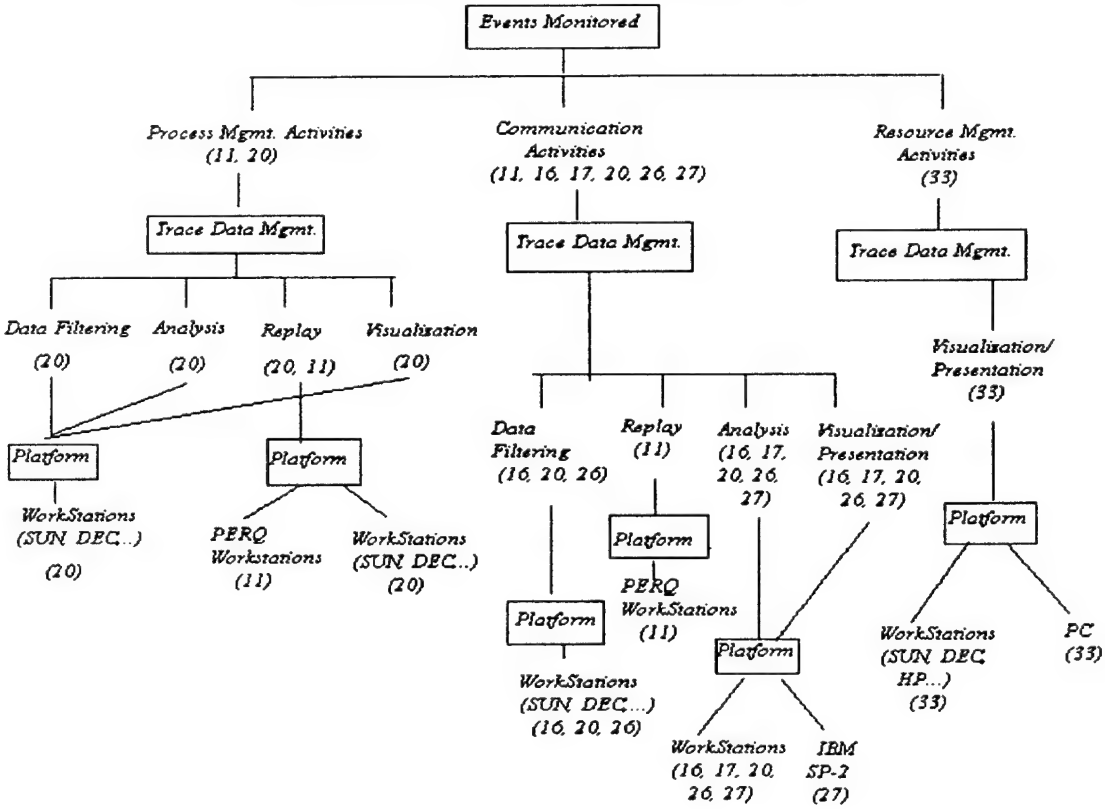


FIGURE 3b (MEASUREMENT TOOLS Contd.)

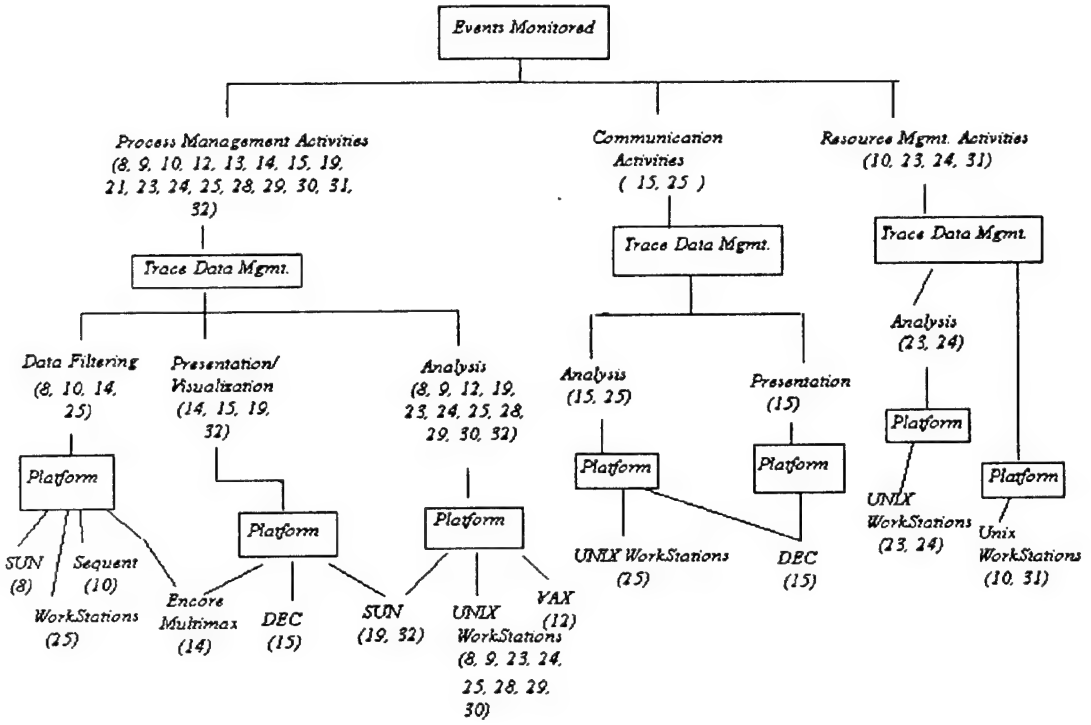


FIGURE 3c
(Measurement Tools Contd.)

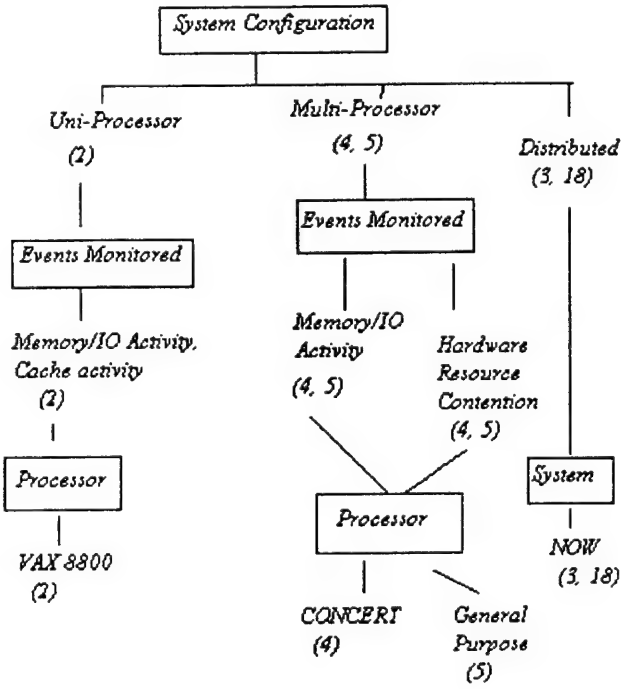


FIGURE 3d
(Measurement Tools Contd.)

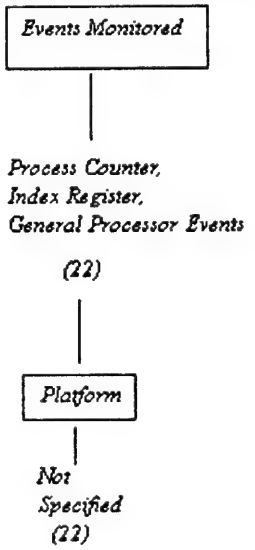


FIGURE 3e
Measurement Tools Contd.

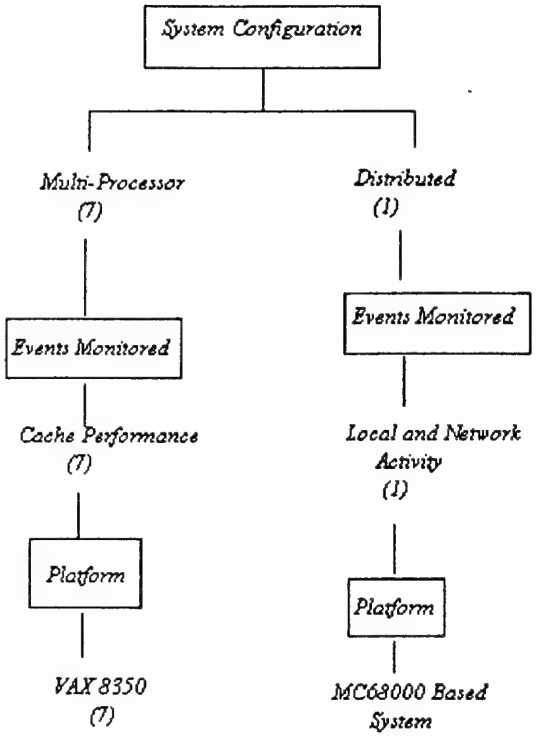
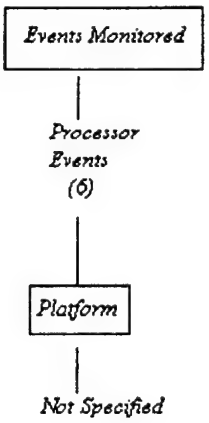
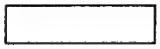


FIGURE 3f
Measurement Tools Contd.



Key :

 Represents a Decision Box.

Measurement Tools List :

(1) *TMP* [220], (2) *Vax 8800* [47], (3) *Zahlmonitor 4* [61], (4) *DLA* [204], (5) *REMS* [43], (6) *TRAMS* [139], (7) *ATUM* [193], (8) *Berkeley UNIX Monitor* [137], (9) *gprof* [89], (10) *Monit* [111], (11) *Radar* [119], (12) *PCA* [62], (13) *Fortran Analyzer* [123], (14) *Parasight* [18], (15) *PSE* [228], (16) *PIE* [120], (17) JADE , (18) INCAS [220], (19) SparcWorks [229], (20) JEWEL [117], (21) SPY [214], (22) TX-2 [148], (23) *MemSpy* [132], (24) *MTOOL* [86], (25) *IPS-2* [138], (26) *Annai* [48], (27) *AIMS* [226], (28) *WAT* [164], (29) *MEDEA* [134], (30) *SP* [140], (31) *Imbench* [229], (32) *SymbIEL* [229], (29) *AXYiOM* [230].

FIGURE 4 (MODELING TOOLS)

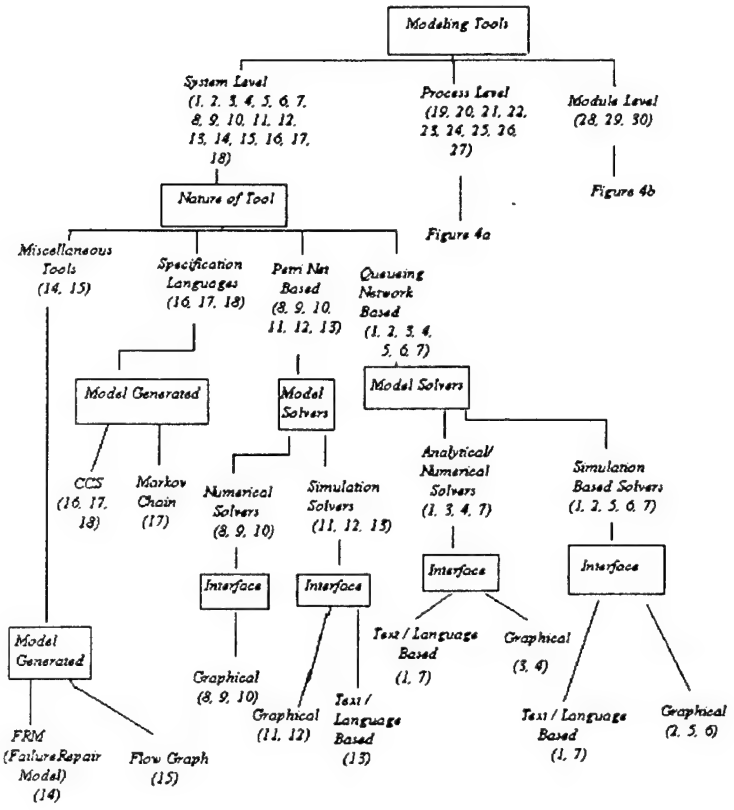


FIGURE 4a (Modeling Tools)

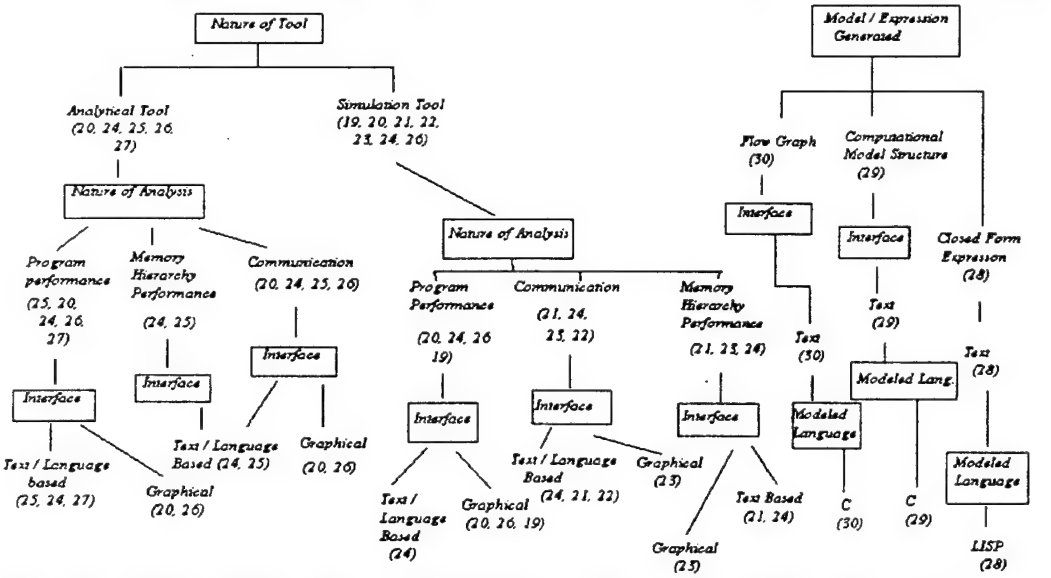


FIGURE 4b (Modeling Tools)

(1) QNAP [212], (2) MODLINE [164], (3) MACOM [114], (4) CAPRES [108], (5) GIST [192], (6) PAW [133], (7) RESQ [127], (8) GreatSPN [46], (9) DSP [122], (10) QPN [25], (11) ADAS [4], (12) Modeler [35], (13) SARA [66], (14) SimPar [94], (15) PARET [150], (16) TIPP [88], (17) PEPA [84], (18) LOTOS [213] (19) Modarch [225], (20) AIMS [226], (21) Axe [221], (22) NET [20], (23) Rice PPT [54], (24) QASE [227], (25) VFCS [45], (26) PEPP [59], (27) MENTOR [60].

8 REFERENCES AND BIBLIOGRAPHY

[1] W. Abu-Sufah and A.Y. Kwok, *'Performance Prediction Tools for Cedar: A Multiprocessor Supercomputer'*, Proceedings of the 12th Annual International Symposium on Computer Architecture, pp. 406-413.

[2] W. Abu-Sufah, D. Kuck, and D. Lawrie, *'Automatic Program Transformations for Virtual Memory Computers'*, In Proc. of the 1979 National Computer Conference, pp. 969-974, June 1979.

[3] L.M. Adama and T.W. Crockett, *'Modeling Algorithm Times on Processor Arrays'*, IEEE Computer (July 1984) pp. 38-43.

[4] *ADAS : An Architecture Design and Assessment System (User Manual)*, Research Triangle Institute, NC, USA (1987).

[5] V.S. Adve et al. *'An Integrated Compilation and Performance Analysis Environment'*, Supercomputing '95, December 1995.

[6] A. Agarwal, *'Performance Tradeoffs in Multithreaded Processors'*, IEEE Transactions on Parallel and Distributed Systems, Vol. 3 No. 5 pp. 525-539, Sept. 1992.

[7] J.R. Agre, M.W. Atkinson, C. Wang, *'Performance Modeling of Distributed Systems through Simulation'*, Proc. Int. Conf. Commun. (1987) pp. 1321-1327.

[8] A. K. Ahuwalia and M. Singhal, *'Performance Analysis of the Communication Architecture of the Connection Machine'*, IEEE Transactions on Parallel and Distributed Systems, Vol. 3, No. 6, pp. 728-738, Nov. 1992.

[9] R.A. Ammar, et al. *'A Computation-Oriented Program Experimentation System (COPES)'*, Proc. IEE Int. Conf. Syst., Man & Cyber. (1989) pp. 861-866.

[10] R.A. Ammar et al. *'An Architecture Assessment Environment for Massively Parallel Computations'*, Proc. IEEE Int. Conf. on Commun. (1987) pp. 1321-1327.

[11] R.A. Ammar and M. Krzych, *'Computer Aided Performance Engineering: A Survey'*, J Comp. Sys. Sci. & Engr, CRL Publ., Ltd., vol 7, no. 3, pp. 170-189, July, 1992.

[12] R.A. Ammar, et al. *'Time Cost Analysis of a Parallel Structure with Multi-communication Nodes in Each Branch'*, 5th Intl. Conf. Par. Dist. Comp. & Sys., Pittsburgh, PA, October, 1992.

[13] R.A. Ammar and P. Zhang, *'A Design and Modeling Methodology for Performance Evaluation in Real-time, Distributed Software Systems'*, IEEE Intl. Conf. Sys. Man & Cyber., Charlottesville, VA, Oct. 13-16, 1991, pp. 707-712.

[14] R.A. Ammar and T. Booth, *'Software Optimization Using User Models'*, IEEE Trans. Systems, Man, & Cyber., vol 18, no 4, Jul/Aug 1988, pp. 552-560.

[15] R.A. Ammar and Q. Bin, *'A Technique to Derive the Detailed Time Costs of Parallel Computations'*, Proc. of the International Computer Software and Applications Conference, Oct. 1988.

[16] R.A. Ammar, S. Ramamurthy and B. Qin, *'Towards Time Analysis of a General Parallel Structure in Shared Memory Environments'*, Proc. of the ISMM International Conference on Parallel and Distributed Computing and Systems, Oct. 1990.

[17] R.A. Ammar and C. Rosiene, *'Visualizing a Hierarchy of Performance Models for Software Systems'*, Software Practice and Experience, Vol. 23, March 1993.

[18] Z. Aral and I. Gertner, *'Non-intrusive and Interactive Profiling in Parasight'*, Proc. ACM/SIGPLAN (1988) pp. 21-30.

[19] D. Attapattu and D. Gannon, *'Building Analytical Models into an Interactive Performance Prediction Tool'*, In Proc. Supercomputing 89, pp. 521-530, Reno, Nevada, 1989, ACM Press.

[20] K. Baclawski, *'A Network Emulation Tool'*, Proc. Symposium on Simulation of Comput. Networks. (1987) pp. 198-206.

[21] R.L. Bagrodia, K.M. Chandy, and J. Misra, *'A Message-based Approach to Discrete-event*

Simulation', IEEE Trans. Softw. Eng., Vol 13 No 6 (June 1987) pp. 654-665.

[22] V. Balasundaram et al., *'Estimating Communication Costs from Data Layout Specifications in an Interactive Data Partitioning Tool'*, Tech. Report, C3p-886, California Institute of Technology, April 1990.

[23] V. Balasundaram, G. Fox, K. Kennedy and U. Kremer, *'A Static Performance Estimator to Guide Data Partitioning Decisions'*, In 3rd ACM Sigplan Symposium on Principles and Practice of Parallel Programming (PPoPP), Williamsburg, VA, April 21-24, 1991

[24] C.C. Barnett, *'Simulation in Pascal with Micro PASSIM'*, Proc. Winter Simulation Conf. (1986) pp. 151-155.

[25] F. Bause and P. Kemper, *'QPNTool for Qualitative and Quantitative Analysis of Queuing Petri Nets'*, in [87] pp. 321-334.

[26] B. Beck and D. Olien, *'A Parallel-Programming Process Model'*, IEEE Software, May 1989, pp. 63-72.

[27] R. Belanger et al. *ModSim User's Manual*, CACI Inc., LaJolla, CA (May 1989).

[28] S. Benker, B. Chapman, anh H. Zima, *'Vienna Fortran 90'*, In Proceedings of the SHPCC Conference 1992, Williamsburg, VA, 1992.

[29] L. Bhuyan, Q. Yang, and D. Agarwal, *'Performance of Multiprocessor Interconnection Networks'*, IEEE Computer, pp. 25-37, Feb. 1988.

[30] B. Biezer, *'Micro Analysis of Computer System Performance'*, Litton Educational Publishing, Inc. 1978.

[31] Q. Bin, H. Sholl, and R.A. Ammar, *'Micro Time Cost Analysis of Parallel Computations'*, IEEE Transactions on Computers, Vol. 40, No. 5, May 1991.

[32] T. Bingman, B. Mackay, M. Schmit, and M. Havira, *'ASAP : A Tool for Analytic Performance Prediction of Software Systems'*, ISCA Conference on Computer Applications in Industry and Engineering, Orlando FL, Dec. 1996.

[33] G. M. Birtwistle, *'Discrete Event Modelling on SIMULA'*, Macmillan, 1979.

[34] R. Blasko, *'Hierarchical Performance Prediction for Parallel Programs'*, Proc. of the International Symposium and Workshop on Systems Engineering of Computer Based Systems, Tucson AZ, March 1995.

[35] A. Blitz et al., *'A General Purpose Modeling and Simulation Tool Exploiting the Petri-net Paradigm'*, Proc. Eastern Simulation Conf. (1988) pp. 21-25.

[36] R. Block, et al., *'Automated Performance Prediction of Message-passing Programs'*, Proc. of the 1995 ACM/IEEE Supercomputing Conference, San Diego CA, Dec. 1995.

[37] P.A. Bobillier, B.C. Kahan, A.R. Probst, *'Simulation with GPSS and GPSS V'*, Prentice Hall, NJ (1976).

[38] J. Bruner, H. Cheong, A. Veidenbaum, and P. Yew, *'Chief : A Parallel Simulation Environment for Parallel Systems'*, In 5th Int'l Parallel Processing Symp.. Anaheim, CA, April 1991.

[39] J. Bruner et al., *'Parallel Computing and the Perfect Benchmarks'*, Tech Report, November, 1991.

[40] A. Carle, et al., *'Automatic Data Layout for Distributed-Memory Machines in the D Programming Environment'*, In Proceedings International Workshop on Automatic Parallelization 1993. Universitat des Saarlandes, Saarbrucken, Germany, March 1993.

[41] B. Carlson, T. Wagner, et al., *'Speedup Properties of Phases in the Execution Profile of Distributed Parallel Programs'*, In Computer Performance Evaluation 1992: Modeling Techniques and Tools, pp. 83-95, 1992. (Ed.) R. Pooley and J. Hillston.

[42] E. Carmona and M. Rice, *'Modeling the Serial and Parallel Fractions of a Parallel Algorithm'*, J Par & Distr Computing 13, 1991, pp. 286-298.

[43] R.J. Carpenter, '*Performance Measurement Instrumentation for Multiprocessor Computers*', Tech. Report NB-SIR 87-3627, Institute for Computer Sciences and Technology, National Bureau of Standards, Aug. 1987.

[44] B.M. Chapman et al., '*Automatic Support for Data Distribution*', pp. 184-199, Springer Verlag, Portland, Oregon, Aug. 1993.

[45] B.M. Chapman, S. Benkner, et al., '*VIENNA FORTRAN Compilation System*: Version 1.0 User's guide'.

[46] G. Chiola, *'A Graphical Petri Net Tool for Performance Analysis'*, Proceedings of the International Workshop on Modeling Techniques and Tools for Performance Evaluation. March 1987, pp. 297-307, AFCET, Paris.

[47] D.W. Clark, P.J. Bannon, and J.B. Keller, *'Measuring VAX 8800 performance with a histogram hardware monitor'*, 15th Ann. Int. Symp. on Computer Arch. (May 1988) pp. 176-185.

[48] C. Clemencon, A. Endo et al., '*Annai : An Integrated Parallel Programming Environment for Multicomputers*', In collection of Tools and Environments for Parallel and Distributed Systems, A. Zaky and T. Lewis (eds.), pp. 33-59, Kluwer Academic Publishers, Feb. 1996.

[49] C. Clemencon, K. Decker et al., '*Tool Supported Parallel Application Development*', Proc. of the IEEE 15th International Conference on Computers and Communications, Scottsdale, AZ, March 1996.

[50] M. Clement and M. Quinn, '*Analytical Performance Prediction on Multicomputers*', Proc. of the IEEE Supercomputing Conference, Nov. 1993.

[51] J. Cohen, *'Computer Assisted Microanalysis of Programs'*, Communications of the ACM, Vol. 25, No. 10, Oct. 1982.

[52] J. Cohen, and A. Weitzman, *'Software Tools for Micro-Analysis of Programs'*, Software - Practice and Experience, Vol. 22 No. 9 pp. 777-808, Sep. 1992.

[53] C.A. Coutant, R.E. Griswold, D.R. Hanson, *'Measuring the Performance and Behavior of icon programs'*, IEEE Trans. Softw. Eng., Vol 9 No 1 (January 1983) pp. 93-102.

[54] R.C. Covington, et al. *'The Rice Parallel Processing Testbed'*, Proc. ACM SIGMETRICS Conf. on Measure. & Modeling of Computer Syst. (1988) pp. 4-11.

[55] R.C. Crain , D.T. Brunner, J.O. Henriksen, *'Advanced Features of GPSS/H'*, Proc. Winter Simulation Conf. (1987) pp. 269-275.

[56] P. Crandall, et al., '*Performance Comparison of Desktop Multiprocessing and Workstation Cluster Computing*', Proc. 5th IEEE International Symposium on High Performance Distributed Computing, Aug. 1996.

[57] P. Crandall and M. Quinn, '*A Decomposition Advisory System for Heterogeneous Data-Parallel Processing*', Proc. of the International Symposium on High Performance Distributed Computing 1994.

[58] F. Darema and J. Prost, '*A Methodology and Tool for Parallel System Performance Analysis*', Proc. 25th Hawaii International Conference on Systems Science, Vol. 2, 1992.

[59] P. Dauphin and A. Quick, *'PEPP : Performance Evaluation of Parallel Programs'*, ITG/GI Fachtagung Messung, Modellierung und Bewertung von Rechen und Kommunikationssystemen, RWTH Aachen, Kurzvortroge and Werkzeugausstellung, pp. 105-108, Sept. 1993.

[60] P. Dauphin and V. Mertsiotakis, *'MENTOR : A Model Based Event Trace Evaluation Support System'*, In the Tools and Posters Proc. of the 7th Int. Conf. on Modeling Techniques and Tools for Computer Performance Evaluation, Vienna, May 1994.

[61] P. Dauphin, R. Hofmann, et al., '*ZM4/SIMPLE: A General Approach to Performance Measurement and Evaluation of Distributed Systems*', Tech. Report, 1/91, University of Erlangen-Nurnberg, IMMD VII, Jan. 1991.

[62] Digital Equipment Corporation, '*VAX Performance and Coverage Analyzer: User's Reference Manual*', 1985. Document Order Number: AA-EB54B-TE.

[63] P. Dubey, M. Flynn, *'Evaluating Performance Tradeoffs Between Fine-grained and Coarse-grained Alternatives'*, IEEE Transactions on Parallel and Distributed Systems, Vol. 6, Jan. 1995.

[64] P. Dubey, A. Krishna, and M. Flynn, *'Analytic Modeling of Multithreaded Pipeline Performance'*, Proc. of the 27th Hawaii International Conference on System Sciences, Wailea HI, Jan. 1994.

[65] D. L. Eager, et al., *'Speedup Versus Efficiency in Parallel Systems'*, IEEE Transactions on Computers, Vol. 38 No. 3 pp. 408-423, March 1989.

[66] G. Estrin et al., '*SARA (Systems ARchitects Apprentice): modeling, analysis, and simulation support for design of concurrent systems'*, IEEE Trans. Softw. Eng., Vol 12 No 2 (Febuary 1986) pp. 293-311.

[67] T. Fahringer, '*Estimating and Optimizing Performance for Parallel Programs'*, IEEE Computer, Vol. 28, Nov. 1995.

[68] T. Fahringer, *'Automatic Cache Performance Prediction in a Parallelizing Compiler'*, In Proc. of the AICA '93 - International Section, Lecce, Italy, Sep. 1993.

[69] T. Fahringer, *'The Weight Finder, An Advanced Profiler for Fortran Programs', In Automatic Parallelization, New Approaches to Code Generation, Data Distribution, and Performance Prediction.'*, Vieweg Advanced Studies in Computer Science, ISBN 3-528-05401-8, Verlag Vieweg, Wiesbaden, Germany, March 1993.

[70] T. Fahringer and H. Zima, *'A Static Parameter based Performance Prediction Tool for Parallel Programs'*, In Invited Paper, Proc. of the 7th ACM International Conference on Supercomputing 1993, Tokyo, Japan, July 1993.

[71] T. Fahringer et al., '*Automatic Performance Prediction to Support Parallelism of Fortran Programs for Massively Parallel Systems'*, In ACM International Conference on Supercomputing 1992, pp. 347-356, Washington D.C., July 1992.

[72] J. Ferrante, V. Sarkar and W. Trash, '*On Estimating and Enhancing Cache Effectiveness'*, In Proc. of the 4th Workshop on Languages and Compilers for Parallel Computing, Santa Clara, CA, Aug. 1991.

[73] D. Ferrari, *'Computer Systems Performance Evaluation'*, Prentice Hall, 1978.

[74] D. Ferrari, and M. Lui, '*A General Purpose Software Measurement Tool'*, Softw. Pract. & Exper., Vol 5 No 2 (April 1975) pp. 181--192.

[75] P.A. Fishwick, '*SIMPACK : Getting Started with Simulation Programming in C and C++'*, Technical Report, University of Florida, July 1992.

[76] E. Foxley and D.J. Morgan, '*Monitoring the Run-time Activity of Algol 68-R Programs'*, Softw. Pract. & Exper., Vol 8 (1978) pp. 29-34.

[77] H. Fromm et al. '*Experiences with Performance Measurement and Modeling of a Processor Array'*, IEEE Trans. Comput., Vol 32 No 1 (January 1983) pp. 15-31.

[78] K. Gallivan et al., '*Performance Prediction of Loop Constructs on Multi-processor Hierarchical-Memory Systems'*, In ACM International Conference on Supercomputing, 1989.

[79] K. Gallivan et al., '*Perfromance Prediction for Parallel Numerical Algorithms'*, International Journal of High Speed Computing, Vol. 3 No. 1 pp. 31-62, 1991.

[80] D. Gannon, D. Atapattu et al., '*A Software Tool for Building Supercomputer Applications'*, Proc. of the Symp. on Parallel Computation and their Impact on Mechanics, Vol 86, pp. 81-94.

[81] R.F. Garcia, '*Simulating with GPSS/PC'*, Proc. Winter Simulation Conf. (1986) pp. 227-234.

[82] H.M. Gerndt, '*Work Distribution in Parallel Programs for Distributed Memory Multiprocessors'*, In Proceedings of the International Conference on Supercomputing, pp. 96-104, Cologne, June 17-21, 1991.

[83] A. Giacalone and S. Smolka, '*Integrated Environment for Formally Well-founded Design*

and Simulation of Concurrent Systems', IEEE Trans. Softw. Eng., Vol 14 No 6 (June 1988) pp. 787-801.

[84] S. Gilmore and J. Hillston, *'The PEPA Workbench: A Tool to Support a Process Algebra based Approach to Performance Modeling'*, in [87], pp. 351-368.

[85] E. Glenebe, 'Performance Analysis of the Connection Machine', Sep. 1989. (Also in ACM SIGMETRICS, 1990).

[86] A. Goldberg and J. Hennessy, *'Performance Debugging Shared Memory Multiprocessor Programs using MTOOL'*, In Proceedings of Supercomputing, pp. 481-490, 1991.

[87] H.A. Goosen et al., *'Experience Using the Chiron Parallel Program Performance Visualization System'*, Tech Report, CS Dept., University of Cape Town, South Africa.

[88] G. Gotz, U. Herzog and M. Rettelbach, *'Multiprocessor and Distributed System Design: The Integration of Functional Specification and Performance Analysis using Stochastic process Algebras'*, in Proceedings of Performance '93, 1993.

[89] S.L. Graham et al., *'gprof: A Call Graph Execution Profiler'*, In Proceedings of the SIGPLAN '82 Symposium on Compiler Construction, June 1982.

[90] M. Gupta and P. Banerjee, *'Compile-time Estimation of the Communication costs on Multicomputers'*, In Proc. Sixth International Parallel Processing Symposium, Beverly Hills, CA, March 1992.

[91] G. Haring and G. Kotsis, Eds., *'Computer Performance Evaluation - Modeling Techniques and Tools'*, 7th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation, Vienna, LNCS 794, Springer-Verlag, May 1994.

[92] F. Hartleb and V. Mertsiotakis, *'Bounds for the Mean Runtime of Parallel Programs'*, Proc. of the 6th Int. Conf. on Modeling Techniques and Tools for Comp. Perf. Evaluation, Editors : R. Pooley and J. Hillston, pp. 143-154, 1992.

[93] K.J. Healy, *'CINEMA tutorial'*, Proc. Winter Simulation Conf. (1986) pp. 207-211.

[94] A. Hein, K. Banch, *'SimPar : A Simulation Environment for Performance and Dependability Analysis of User-defined Fault Tolerant Parallel Systems'*, Internal Report 1/95, IMMD III, January 1995.

[95] A. Hein, K.K. Goswami, *'Combined Performance and Dependability Evaluation with Conjoint Simulation'*, Proceedings of the ESS '95, European Simulation Symposium, Erlangen, Germany, October 1995.

[96] J.C. Hickey, H. Hotta, T. Petitjean, *'Computer-Assisted Microanalysis of Parallel Programs'*, ACM Transactions on Programming Languages and Systems, Vol. 14, No. 1, Jan. 1992.

[97] R. Hockney, *'A Framework for Benchmark Performance Analysis'*, Tech. Report, Dept. of Electronics and Computer Sc., 1992.

[98] M. Holliday, M. Strumm, *'Performance Evaluation of Hierarchical Ring-based Shared Memory Multiprocessors'*, IEEE Transactions on Computers, Vol. 43, No. 1, Jan. 1994.

[99] C. Hrischuk, J. Rolia, and C. Woodside, *'Automatic Generation of a Software Performance Model using an Object-oriented Prototype'*, Proc. of the IEEE International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Durham, NC, Jan. 1995.

[100] J. Hsu and P. Banerjee, *'Performance Measurement and Trace Driven Simulation of Parallel CAD and Numerical Applications on a Hypercube'*, IEEE Transactions on Parallel and Distributed Systems, Vol. 3, July 1992.

[101] O. Ibe, H.Choi, and K. Trivedi, *'Performance evaluation of Client-Server Systems'*, IEEE transactions on Parallel and Distributed Systems, Vol. 4, No. 11, Nov. 1993.

[102] R. Jain, *'The Art of Computer Systems Performance Analysis'*, Wiley Professional Computing, 1991.

[103] J. Joyce, et al., '*Monitoring Distributed Systems*', ACM Trans. Comput. Syst., Vol. 5, pp. 121-150, May 1987.

[104] C. Juiz and R. Puigjaner, '*Approximate Performance Models of Real-time Software Systems*', Proc. of the 3rd IEEE International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, Durham NC, Jan. 1995.

[105] J.R. Jump, '*The YACSIM Reference Manual*', Rice University, 1993.

[106] A. Kapelnikov, et al., '*A Modeling Methodology for the Analysis of Concurrent Systems and Computations*', J par & Distr Computing, Jun. 1989, pp. 568-597.

[107] A. Karp and H. Flatt, '*Measuring Parallel Processor Performance*', CACM, Vol. 33, No. 5, May 1990, pp. 539-543.

[108] M.P. Kastner, K.R. Pattipati and S.R. Dunham, '*CAPRES: a Software Tool for Modeling and Analysis of Fault-tolerant Computer Architectures*', Proc. Int. Conf. Syst. Man & Cybern., 1989.

[109] R. Katti, '*Performance Analysis of Parallel Computations*', Proc. of the ISCA International Conference on Parallel and Distributed Computing Systems, Oct. 1993.

[110] K. Kennedy et al., '*Static Performance Estimation in a Parallelizing Compiler*', Tech. Report TR91-174, Dept. of Computer Sc., Rice University, Dec. 1991.

[111] T. Kerola and H. Schwetman, '*Monit: A Performance Monitoring Tool for Parallel and Pseudo-Parallel Programs*', In Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, May 1987.

[112] C. Kesselman, '*Tools and Techniques for Performance Measurement and Performance Improvement in Parallel Processing*', UCLA Tech. Report, UCLA-CS-TR-91-03, 1991.

[113] V. Knapp, '*The Smalltalk Simulation Environment*', Proc. Winter Simulation Conf. (1986) pp. 125-128.

[114] U.R. Krieger, B. Muller-Clostermann and M. Sczittnick. '*Modeling and Analysis of Communication Systems Based on Computational Methods for Markov Chains*', IEEE Journal on Selected Areas in Communications, Vol. 8, No. 9, pp. 456-470, 1990.

[115] C. Krusal and M. Snir, '*The Performance of multistage Interconnection Networks for Multiprocessors*', IEEE Transactions on Computers, C-32, pp. 1091-1098, Dec. 1983.

[116] P. Krystosek, S. Sirazi and G. Campbell, '*REN: A Reconfigurable Experimental Network*', Proc. Symposium on Simulation of Comp. Networks, August 1987 pp. 45-50.

[117] F. Lange et al., '*JEWEL: Design and Implementation of a Distributed Measurement System*'.

[118] E. Lazowska, J. Zahorjan, G. Graham, K. Sevcik, '*Quantitative System Performance*', Prentice Hall Inc., 1984.

[119] R.J. LeBlanc and A.D. Robbins, '*Event Driven Monitoring of Distributed Programs*', In Proceedings of the Fifth International Conference on Distributed Computing Systems, May 1985.

[120] T. Lehr, D.L. Black, '*Mach Kernel Monitor with application using the PIE environment*'.

[121] J. Li and M. Chen, '*Compiling Communication-efficient Programs for Massively Parallel Machines*', IEEE Transactions on Parallel and Distributed Systems, Vol. 2 No. 3 pp. 361-376, July 1991.

[122] C. Lindemann, '*DSPNExpress: A Software Package for the Efficient Solution of Deterministic and Stochastic Petri Nets*', 7th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation, Vienna, LNCS 794, Springer-Verlag, May 1994, pp. 9-20.

[123] G. Lyon and R. Stillman, '*Simple Transforms for Instrumenting FORTRAN Decks*', Softw. Pract. & Exper., Vol 5 No 4 (October 1975) pp. 347-358.

[124] N.B. MacDonald, *'Predicting the Execution Time of Sequential Scientific Codes'*, In Proceedings of International Workshop on Automatic Parallelization 1993, Universitat des Saarlandes, Saarbrucken, Germany, March 1993.

[125] M.H. MacDougall, *'Simulating Computer Systems : Techniques and Tools'*, MIT Press, 1987.

[126] B. Mackay and H.A. Sholl, *'Communication Alternatives for a Distributed Real-Time System'*, Proc. of the ISCA Computer Applications in Industry and Engineering Conference, Honolulu HI, Nov. 1995.

[127] E.A. MacNair, *'An Introduction to the Research Queueing Package'*, Proc. Winter Simulation Conf, 1985.

[128] S. Madala and J. Sinclair, *'Performance of Synchronous Parallel Algorithms with Regular Structures'*, IEEE Trans Par & Distr Sys, Vol. 2, No. 1, Jan. 1991, pp. 105-116.

[129] B. Malloy and M.L. Soffa, *'Simcal:The Merger of Simula and Pascal'*, Proc. Winter Simulation Conf. (1986) pp. 397-403.

[130] E. Manchon, *'The PEPS Modeling Tools'*, Proc. of the Performance Evaluation of Parallel Systems Workshop, ESPRIT Project, Univ. of Warwick, UK, Nov. 1993.

[131] M.A. Marson et al., *'Stochastic Petri Nets as a Tool for the Analysis of High-Performance Distributed Architectures'*, Chapter 12, pp. 578-613, Van Nostrand Reinhold, New York, 1990, (Ed.) S.P. Kartashev and S.I. Kartashev.

[132] M. Martonosi, A. Gupta, et al., *'Memspy : Analyzing Memory System Bottlenecks in Programs'*, In the Proceedings of Sigmetrics '92, pp. 1-12, ACM, 1992.

[133] B. Melamed and R.J.T. Morris, *'Visual Simulation: The Performance Analysis Workstation'*, Computer, August 1985.

[134] A. Merlo, and P. Worley, *'Analysing PICL Trace Data with MEDEA'*, in [87] pp. 445-464.

[135] H. Mierendorff et al., *'Performance Estimates for SUPRENUM System'*, Parallel Comput. 7 (1988), pp. 357-366.

[136] H. Mierendroff and R. Schwarzwald, *'LAPAS: A Performance Evaluation Tool for Large Parallel Systems'*, In Tagungsband, Munchen, Marz, 1990, ITG/GI Fachtagung.

[137] B.P. Miller, C. MacRander, and S. Sechrest, *'A Distributed Programs Monitor for Berkeley Unix'*, Softw. Pract. & Exper., Vol 16 No 2 (Febuary 1986) pp. 183-200.

[138] B.P. Miller, et al. *'IPS-2: The Second Generation of a Parallel Program Measurement System'*, IEEE Trans. Parallel Distributed Syst., Vol. 1, no. 2, pp. 206-217, Apr. 1990.

[139] A. Mink, et al., *'Hardware-assisted Multiprocessor Performance Measurements'*, Tech. Report NBSIR 87-3585, Institute for Computer Sciences and Technology, National Bureau of Standards, June 1987.

[140] C. Minkowitz, V. Vetland and P.H. Huges, *'A Modular Approach to System Structure and Specification'* in [93], pp. 83-86.

[141] P. Mohapatra, et al., *'Performance Analysis of Cluster-Based Multiprocessors'*, IEEE Transactions on Computers, Vol. 43, No. 1, Jan. 1994.

[142] B. Mohr, *'SIMPLE: A Performance Evaluation Tool Environment for Parallel and Distributed Systems'*, in Distributed Memory Computing, 2nd European Conference, EDMCC2 (A. Bode, ed.), pp. 80-89, April 1991.

[143] M.F. Morris, P.F. Roth, *'Computer Performance Evaluation : Tools and Techniques for Effective Analysis'*, Van Nostrand Reinhold Company, NY, 1982.

[144] W.E. Nagel und A, Arnold, *'PARvis: Ein Werkzeug Zur Visualisierung Von Parallelen Prozessen Auf Mehrprozessorsystemen'*, Proc. 7 ITG/Gi Fachtagung MMB '93, pp. 178-187, 1993.

[145] J. Nehmer et al., *'Key Concepts of the INCAS Multicomputer Project'*, TOSE, IEEE, Vol. SE-13, No. 8, Aug. 1987, 913-923.

[146] M. Neilforoshan, R. Ammar et al., *'Optimizing the Time Cost of Parallel Structures by Scheduling Parallel Processes to Access*

Critical Sections', University of Connecticut, Booth Research Center, Tech. Report #TR-92-02, 1992.

[147] R. Nelson. '*A Performance Evaluation of a General Parallel Processing Model'*, Performance Evaluation Review, Vol. 18, No. 1, Aug. 1991, pp. 52-60.

[148] A.G. Nemeth and P.D. Rovner, *'User Program Measurement in a Time Shared Environment'*, Commun. ACM, Vol 14 No 10 (October 1971) pp. 661-666.

[149] *Network II.5 User's Manual*, CACI Inc., Los Angeles CA (1987).

[150] K.M. Nichols and J.T. Edmark, '*Modeling Multicomputer Systems with PARET'*, Computer (May 1988) pp. 39-48.

[151] G.R. Nudd, et al., '*A Layered Approach to the Characterization of Parallel Systems for Performance Prediction'*, Proc. of the Performance Evaluation of Parallel Systems Workshop, ESPRIT Project, Univ. of Warwick, U.K., Nov. 1993.

[152] R.M. O'Keefe, and R.M. Davies, *'Discrete Visual Simulation with Pascal_SIM'*, Proc. Winter Simulation Conf. (1986) pp. 517-529.

[153] J.J. O'Reilly, '*SLAM II, a Tutorial'*, Proc. Winter Simulation Conf. (1986) pp. 60-65.

[154] C. Pancake, '*Software Support for Parallel Computing: Where are We Headed ?'*, CACM, Vol. 34, No. 11, Nov. 1991, pp. 52-64.

[155] C. Pancake et al., '*Performance Evaluation Tools for Parallel and Distributed Systems'*, IEEE Parallel and Distributed Technology, Vol. 3, No. 4, Winter 1995.

[156] E. Papaefstathiou et al., '*An Overview of the CHIPS Performance Prediction Toolset for Parallel Systems'*, Proc. of the ISCA Conference on Parallel and Distributed Computing Systems, 1995.

[157] E. Papaefstathiou et al., '*A Layered Approach to Parallel Software Performance Prediction: A Case Study'*, In Massively Parallel Processing Applications and Development, Elsevier Science, New York, 1994.

[158] N. Patel, '*Structuring Analytical Performance Models using Mathematica'*, Proc. of the International Conference on Modeling Techniques and Tools for Computer Performance Evaluation, Edinburgh, Sept. 1992.

[159] D. Pease, A. Ghafoor, et al., '*PAWS: A Performance Evaluation Tool for Parallel Computing Systems'*, IEEE Computer. Vol. 24, Jan. 1991.

[160] C.D. Pegden, '*Introduction to SIMAN'*, Proc. Winter Simulation Conf. (1986) pp. 95-103.

[161] J.P. Penny, P.J. Ashton, and A.L. Wilkinson, '*Data Recording and Monitoring for Analysis of System Response times'*, Computer J., Vol 29 No 5 (October 1986) pp. 396-403.

[162] The Perfect Club, '*The Perfect Club Benchmarks: Effective Performance Evaluation of Supercomputers'*, International Journal of Supercomputing Applications, 1989.

[163] R. Pooley, '*A Survey of Perfromance Analysis Tools in Europe'*, CSG Report Series, ECS-CSG-12-95, Computer Systems Group, University of Edinburgh, July 1995.

[164] R. Pooley, '*The Integrated Modeling Support Environment'*, 5th International Conference on Modeling Techniques and Tools for Computer Performance Evaluation, Torino, Elsevier, 1992, pp.1-16.

[165] J.P. Prost and M. Becker, '*Modeling Methodology for Performance Evaluation of Parallel Architectures'*, Intl. J of High Speed Comp., Vol. 1, No. 4, Dec 1989, pp. 563-601.

[166] J.P. Prost and S. Kipnis, '*A Multilevel Trace-Driven Simulation Approach for Performance Analysis of Distributed Memory Programs'*, RC 17612 (#77650), IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10598, Jan. 1992.

[167] B. Qin, H.A. Sholl et al, '*Micro Time Cost Analysis of Parallel Computations'*, IEEE Trans Comp., Vol. 40, No. 5, pp. 613-628, May 1991.

[168] B. Qin, H.A. Sholl and R.A. Ammar, '*OPAS: A Tool to Minimize the Time Cost of*

Parallel Computations Through Optimal Processing Power Allocation', Softw. Pract. & Exper., Vol 20 No 3 (March 1990) pp. 283-300.

[169] B. Qin and H.A. Sholl, *'DFAS: A Tool to Analyze the Time Cost of Data Flow Programs'*, ISMM Int. Conf. on Comput. Applicatons. in Design Simulation & Analysis, Nevada (Feb 1989) pp. 64-67.

[170] J. Ramanujam and P. Sadayappan, *'Compile-time Techniques for Data Distribution in Distributed Memory Machines'*, IEEE Transactions on Parallel and Distributed Systems, Vol. 2 No. 4 pp. 472-482, Oct. 1991.

[171] B. Ramkumar and G. Chillariga, *'Performance Prediction for Portable Parallel Execution on MIMD Architectures'*, Proc. of the IEEE International Parallel Processing Symposium, Santa Barbara CA, April 1995.

[172] C. Rathbone, *'Processing Time Analysis in a Distributed/ Parallel Environment with Interrupts'*, Masters Thesis, University of Connecticut, 1989.

[173] S.D. Roberts, *'Modeling and Simulation with INSIGHT'*, Proc. Winter Simulation Conf. (1986) pp. 104-112.

[174] M. Rosenblum, et al., *'Complete Computer System Simulation: The SimOS Approach'*, IEEE Parallel and Distributed Technology, Vol.3, No. 4, Winter 1995.

[175] C. Rosience, and R.A. Ammar, *'Data Modeling Framework for Performance Analysis of Sequential and Parallel Software'*, Proc. of the 21st Annual Computer Science Conference, Indianapolis IN, Feb. 1993.

[176] G. Ries et al., *'DEPEND: A Simulation Environment for System Dependability Modeling and Evaluation'*, Proceedings of the 2nd IEEE Int. Comp. Perf. and Dependability Symposium (IPDS '96), Sept. 1996.

[177] S. Ruiz-Mier, J. Talavage, D. Ben-Arieh, *'Towards a Knowledge-based Network Simulation Environment'*, Proc. Winter Simulation Conf., (1985) pp. 141-150.

[178] R. Saavedra, *'CPU Performance Evaluation and Execution Time Prediction using Narrow Spectrum Benchmarking'*, Ph.D. thesis, University of California at Berkeley, 1992.

[179] M.L. Samuels and J.R. Spiegel, *'The Flexible Ada Simulation Tool (FAST) and its extensions'*, Proc. Winter Simulation Conf. (1987) pp. 175-184.

[180] V. Sarkar , *'Determining Average Program Execution Times and their Variance'*, Proc. of the Conference on Programming Languages Design and Implementation, June 1989.

[181] V. Sarkar, *'Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors'*, Research Monograph in Parallel and Distributed Computing, Pitman 1989.

[182] V. Sarkar, *'Automatic Partitioning of a Program Dependence Graph into Parallel Tasks'*, IBM J. of Research and Dev., September/November 1991.

[183] C.H. Sauer,E.A. MacNair, and S. Salza, *'A Language for Extended Queuing Models'*, IBM J. Res. & Dev., Vol 24 No 6 (November 1980).

[184] H. Schwetman, 'CSIM: a C-based, Process-oriented Simulation Language', Proc. Winter Simulation Conf. (1986) pp. 387-396.

[185] T.J. Schriber, *'Introduction to GPSS'*, Proc. Winter Simulation Conf. (1986) pp 75-78.

[186] M. Schumann, *'Efficient Performance Prediction for Parallel Programs'*, Ph.D. Dissertation, Technical University of Munich, Germany, May 1996.

[187] M. Schumann, *'Automatic Performance Prediction to Support Cross Development of Parallel Programs'*, Proc. of the ACM Symposium on Parallel and Distributed Tools, May 22-23, 1996.

[188] H.A. Sholl, R.A. Ammar, and B.Quinn, *'TCAS: A Time Cost Analysis System for Parallel Computations'*, ISMM Int. Conf. on Mini and Microcomput., Miami Beach, FL (December 1988) pp. 217-220.

[189] H.A. Sholl, R.A. Ammar and C. Xu, *'Performance Analysis of Software Designs*

Influenced by Cache Memory', International Journal of Science and Technology, Vol. 6 No. 1, Spring 1993.

[190] H.A. Sholl and S. Kim, '*An Approach to Performance Modeling as an Aid in Structuring Real-Time Distributed System Software'*, Proc. 19th Hawaii International Conference on System Science, Jan, 1986.

[191] *Simscript II.5 Programming Language*, CACI Inc., Los Angeles, CA (1987).

[192] Sinclair, J B, Doshi, K A and Madala, S, '*Computer Performance Evaluation with GIST: A Tool for Specifying Extended Queuing Network Models'*, Proc. Winter Simulation Conf., 1989.

[193] R.L. Sites, and A. Agarwal, '*Multiprocessor Cache Analysis Using ATUM'*, In Fifteenth International Symposium on Computer Architecture, June 1988.

[194] D.L. Smith, '*Performance Analysis of Software for an MIMD Computer'*, Proc. of ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, 1982.

[195] L.W. Smith, '*Software Performance Engineering: A Case Study Including Performance Comparison with Design Alternatives'*, IEEE Transactions on Software Engineering, Vol 19, No. 7, July 1993.

[196] C. Smith, B. Wong, '*SPE Evaluation of a Client/Server Application'*, Proc. of the 20th International Conference for the Resource Management and Performance Evaluation of Enterprise Computing Systems, Orlando, Fl, Dec. 1994.

[197] C.U. Smith, *'Performance Engineering of Software Systems'*, The SEI Series in Software Engineering, Addison-Wesley Publishing Co., 1990.

[198] F. Sotz, '*A Method for Performance Prediction of Parallel Programs'*, Proceedings of the Joint Int. Conf. on Vector and Parallel Processing, pp. 98-107, Sept. 1990.

[199] K. Suzuke, A. Sangiovanni-Vincentelli, '*Efficient Software Performance Estimation Methods for Hardware/Software Codesign'*, Proc. of the 33rd Design Automation Conference, Las Vegas NV, June 1996.

[200] C.R. Standridge, et al., '*A Tutorial on TESS: The Extended Simulaton System'*, Proc. Winter Simulation Conf. (1986) pp. 212-217.

[201] K.L. Stanwood, L.N. Waller, and G.C. Marr '*System Iconic Modeling Facility'*, Proc. Winter Simulation Conf. (1986) pp 531-536.

[202] J.T. Stasko, '*The PARADE Environment for Visualizing Parallel Program Execution : A Progress Report'*, Tech Report, GVU, Georgia Institute of Technology, Atlanta, January 1995.

[203] P. Stenstrom, et al., '*Shared Data Structures in a Distributed System - Performance Evaluation and Practical Considerations'*, In Proc. International Seminar on Performance of Distributed and Parallel Systems, Kyoto, Japan, Dec. 1988.

[204] T.L. Sterling, et al., '*Multiprocessor Performance Measurement Using Embedded Instrumentation'*, In Proceedings of the 1988 International Conference on Parallel Processing, 1988.

[205] B. Stramm and F. Berman, '*Predicting the Performance of Large Programs on Scalabel Multi-computers'*, In Proceedings of the Scalable High Performance Computing Conference SHPCC-92, Williamsburg, VA, April 1992.

[206] A. Sussman, '*Model Driven Mapping of Computation onto Distributed Memory Parallel Computers'*, Ph.D. thesis, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA, 1991.

[207] H. Tokuda and M. Kotera, '*A Real-time Toolset for the ARTS Kernel'*, Carnegie Mellon Univ., Tech. Report, CMU-CS-88-180, Sept. 1988.

[208] J. Torrellas, J. Hennessy et al., '*Analysis of Critical Architectural and Program Parameters in a Hierarchical Shared-Memory Multiprocessor'*, Performance Evaluation Review, Vol. 18, No. 1, May 1990, pp. 163-172.

[209] R.D. Trammel, '*The Big Picture: Visualizing System Behaviour in Real Time'*, in

Proc. 1990 USENIX Summer Conf., Anaheim, CA, June 1990, pp. 257-266.

[210] F.A. Van-Catledge, *'Towards a General Model for Evaluating the Relative Performance of Computer Systems'*, Intl J Supercomput Appl, Vol. 3, No. 2, Summer 1989, pp. 100-108.

[211] J. Veenstra and R. Fowler, *'MINT: A Front End for Efficient Simulation of Shared Memory Multiprocessors'*, Proc. of the IEEE International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Durham NC., Jan. 1995.

[212] M. Veran and D. Potier, *'QNAP 2: A Portable Environment for Queuing System Modeling'*, Proceedings of Modeling Techniques and Tools for Computer Performance Evaluation', North Holland, 1985, pp. 25-63.

[213] A. Valderruten, O. Hjiej, et al., *'Deriving Queuing Networks Performance Models from Annotated LOTOS Specifications'*, in [93], pp. 120-130.

[214] W.M. Waite, *'A Sampling Monitor for Application Programs'*, Softw. Pract. & Exper., Vol 3 No 1 (January 1973) pp. 75-79.

[215] K. Wang, *'Precise Compile-time Performance Prediction for Superscalar-based Computers'*, Proc. of the Conference on Programming Language Design and Implementation, June 1994.

[216] S. Wathne, H.A. Sholl and R.A. Ammar, *'Task Partitioning of Multi-channel, Distributed Real-Time Systems'*, Proc. of the ISCA Computer Applications in Industry and Engineering Conference, Honolulu HI, Nov. 1995.

[217] B. Wegbreit, *'Mechanical Program Analysis'*, Commun. ACM, Vol. 18 No. 9 (September 1975), pp. 528-539.

[218] R. Williams and W.E. Nagel, *'Optimization of Output Bandwidth from a Paragon'*, Tech. Report CCSF-44, Caltech Concurrent Supercomputing Facilities, 1994.

[219] C. Woodside, *'Three-view Model for Performance Engineering of Concurrent Software'*, IEEE Transactions on Software Engineering, Vol. 21, No. 9, Sept. 1995.

[220] D. Wybranietz and D. Haban, *'Monitoring and Performance Measuring Distributed Systems during Operation'*, Proc. ACM SIGMETRICS Conf. on Measure. & Modeling of Comp. Sys. (1988) pp. 197-206.

[221] J.C. Yan and S.F. Lundstrom, *'Axe : A Simulation Environment for Actor-like Computations on Ensemble Architectures'*, Proc. Winter Simulation Conf. (1986) pp. 424-429.

[222] C.Q. Yang and B. Miller, *'Performance Measurement for Parallel and Distributed Programs: A Structured and Automatic Approach'*, IEEE Trans Software Engg., Vol. 15, No. 12, Dec. 1989, pp. 1615-1629.

[223] X. Zhang and X. Qin, *'Performance Prediction and Evaluation of Parallel Processing on a NUMA Multiprocessor'*, IEEE Transactions on Software Engineering, Vol. 17, Oct. 1991.

[224] More information about SIM++ can be obtained from the Institute of Telematics, University of Karlsurhe.

[225] More information can be obtained from : www.simulog.fr/US/html/prods/modarch.html.

[226] More information on the tool can be obtained from : www.nas.nasa.gov/NAS/Tools/Projects/AIMS

[227] More information about the product can be obtained by writing to : info@advsystech.com

[228] More information about the product can be obtained from : www.digital.com/info/hpc/f90/pse.html

[229] More information about the tool can be obtained at : www.eu.sun.com/95091/tuning/tools.html

[230] For more information about this tool, visit: www.ov.com/text/products/perf_trend.html

[231] More Information about the tool can be obtained at : www.almaden.ibm.com/watson/pv

[232] More Information about this tool can be obtained at :
www.intel.com

References List for Statistical Support Tools:

[233] *Catalyst.* Sun Microsystems, Inc., Mountain View, CA (1988).

[234] *PC Magazine.* Vol. 8 No. 5 (March 1989).

[235] A. M. Law, S. G. Vincent, "*UniFit User's Guide*", Simulation, Modeling and Analysis Co., Tucson, AZ (1985).